REIHE AUTOMATISIERUNGSTECHNIK · BAND 140



Lutz Kern
Klaus Ober
Jörg Schumann

ESER – Programmierung im Betriebssystem DOS/ES

Systembeschreibung, Assemblerprogrammierung

REIHE AUTOMATISIERUNGSTECHNIK

* Zur Zeit lieferbare oder in Vorbereitung befindliche Bände

*RA 1 *Schwarze:* Grundbegriffe der Automatisierungstechnik
RA 2 *Gottschalk:* Bauelemente der elektrischen Steuerungstechnik
RA 3 *Berg:* Einführung in die Hydraulik
RA 4 *Schöpflin:* Netzregelungen
*RA 5 *Schubert:* Digitale Kleinrechner
*RA 6 *Sydow:* Elektronische Analogrechner
RA 7 *Götte:* Elektronische Bauelemente in der Automatisierungstechnik
RA 8 *Bojartschenko/Schinjanski:* Magnetische Verstärker
*RA 9 *ten Brink/Kauffold:* Entwurf und Ausführung von Steueranlagen
RA 10 *Schwarze:* Regelkreise mit I- und P-Reglern
RA 11 *Peschel:* Regelkreise mit PID-Reglern
RA 12 *Stuchlik:* Programmgesteuerte Universalrechner
*RA 13 *Kautsch:* Elektrische Meßverfahren für nichtelektrische Größen
RA 14 *Ehrhardt:* Fernsteuerung
RA 15 *Schöpflin:* Projektierung von Regelungsanlagen
RA 16 *Lüdtke:* Betriebserfahrungen mit einer automatischen Großanlage
*RA 17 *Schroedter/Meyer:* Betriebsmeßwesen
RA 18 *Fritzsch:* Grundlagen der elektrischen Antriebsregelung
RA 19 *Ahner/Bode:* Elektronische Datenverarbeitung in der Ökonomie
RA 20 *Dittmann:* Kennwertermittlung von Regelstrecken und Regelgeräten
RA 21 *Fuchs:* Digitale Regelungen
RA 22 *Borgwardt:* Gasanalysen-Meßtechnik
RA 23 *Finger:* Elektrische Wägetechnik
RA 24 *Obenhaus:* Fernmeßeinrichtungen
*RA 25 *Bär:* Einführung in die Schaltalgebra
RA 26 *Borgwardt:* Flüssigkeitsanalysen-Meßtechnik
RA 27 *Liebers:* Temperaturmessungen
*RA 28 *Hummitzsch:* Zuverlässigkeit von Systemen
*RA 29 *Berg:* Hydraulische Bauelemente in der Automatisierungstechnik
*RA 30 *Peschel:* Kybernetik und Automatisierung
RA 31 *Schroedter:* Standmessung in Behältern
RA 32 *Meyer:* Volumen- und Durchflußmessung von Flüssigkeiten und Gasen
RA 33 *Hartmann:* Regelkreise mit Zweipunktreglern
RA 34 *Roeber:* Meßeinrichtungen
*RA 35 *Wagner:* Automatisierungstechnik — Einführung und Überblick
*RA 36 *Zemlin:* Grundzüge des Frequenzkennlinienverfahrens
*RA 37 *Berg:* Anwendung der Hydraulik in der Automatisierungstechnik
*RA 38 *Gottschalk:* Elektronische Bausteinsysteme der Digitaltechnik
RA 39 *Wolff:* Anwendung des Frequenzkennlinienverfahrens
*RA 40 *Bär/Fuchs:* Kleines Lexikon der Steuerungs- und Regelungstechnik
RA 41 *Greif:* Anwendung lichtelektrischer Empfänger
RA 42 *Bär:* EDV — Grundstufe der COBOL-Programmierung
RA 43 *Bär:* EDV — Oberstufe der COBOL-Programmierung
RA 44 *Bär:* EDV — Praxis der COBOL-Programmierung
RA 45 *Bittner:* Pneumatische Funktionselemente
RA 46 *Fuchs/Könitzer:* Digitale Meßwerterfassung
RA 47 *Andersen:* ALGOL 60 — eine Sprache für Rechenautomaten
RA 48 *Götte:* Feuchtemeßtechnik
RA 49 *Gena:* Automatisierung der chemischen Industrie
*RA 50 *Schwarze:* Regelungstechnik für Praktiker

140

REIHE AUTOMATISIERUNGSTECHNIK

Herausgegeben von
G. Brack, H. Fuchs, G. Paulin, R. Piegert, G. Schwarze und E.-G. Woschni

ESER – Programmierung im Betriebssystem DOS/ES

Systembeschreibung, Assemblerprogrammierung

Lutz Kern
Klaus Ober
Jörg Schumann



VEB VERLAG TECHNIK BERLIN

Federführender Herausgeber: *Gerhard Paulin*
Begutachtender Herausgeber: *Gunter Schwarze*
Lektor: *Jürgen Reichenbach* Einbandgestaltung: *Kurt Beckert*
VT 3/6/4898 ES 20 K 2 DK 681.322:631.3.05/.06

VLN 201 Dg.-Nr. 370/51/73 Deutsche Demokratische Republik
Gesamtherstellung: Reyhérdruck Gotha


Eingetragene Schutzmarke des Warenzeichenverbandes
Regelungstechnik e. V., Berlin

Bestellnummer: 552 070 4

Vorwort

In der DDR wird gegenwärtig der Einsatz der Modelle des Einheitlichen Systems der Elektronischen Rechentechnik (ESER) R20 (ES 1020) und R40 (ES 1040) vorbereitet. Außerdem gelangt die in der DDR entwickelte elektronische Datenverarbeitungsanlage (EDVA) ROBOTRON 21, die mit den geräte- und anwendungstechnischen Systemeigenschaften des ESER voll übereinstimmt, zum Einsatz. Durch die Bereitstellung des Betriebssystems DOS/ES kann die moderne Gerätetechnik entsprechend ihren Möglichkeiten optimal genutzt werden.

Im Rahmen der REIHE AUTOMATISIERUNGSTECHNIK soll in einer Folge von inhaltlich abgestimmten Bänden ein Überblick über die Arbeit mit dem Betriebssystem DOS/ES gegeben werden. Im einzelnen werden folgende Themen behandelt:

Systembeschreibung, Assemblerprogrammierung (RA 140)

Ein- und Ausgabetechnik, Arbeit mit Magnetplattenspeichern (RA 141)

Steuerung der Programmabarbeitung, Bibliotheksführung, Hilfsprogramme (RA 142)

Dateibehandlung; Programmierbeispiele (RA 143)

Der vorliegende Band wendet sich an Leser, die sich einen Überblick über das ESER-System und die EDVA ROBOTRON 21 unter dem Gesichtspunkt des Gerätespektrums und der Assemblerprogrammierung verschaffen wollen. Erfahrungsgemäß macht das Einarbeiten in einen Maschinenkode nach vollständigen Systemhandbüchern, die nicht unter methodischen Gesichtspunkten geschrieben wurden, erhebliche Schwierigkeiten. Deshalb betrachten wir es als ein wichtiges Anliegen dieser Bände, den Zugang zu vollständiger bzw. spezieller Systemliteratur zu erleichtern, wie sie etwa durch [5] und [6] gegeben ist.

Es ist nicht möglich, die gesamte Assemblerprogrammierung zu behandeln. Der gebotene Stoff stellt eine Auswahl wesentlicher Elemente der Assemblerprogrammierung dar. Komplexe Programmierübungen werden vorwiegend in RA 143 behandelt.

Für das Verständnis des dargelegten Stoffes ist es günstig, wenn der Leser über den grundsätzlichen Aufbau und die Arbeitsweise von EDVA informiert ist sowie Grundlagen der Programmiertechnik kennt.

Die Autoren danken dem Herausgeber, Herrn *G. Paulin*, für die freundlichen Hinweise zur Gestaltung der Bände. Dem VEB Verlag Technik sind wir für die Aufnahme der Bände in die REIHE AUTOMATISIERUNGSTECHNIK dankbar.

Leipzig

Die Autoren

Inhaltsverzeichnis

1. ESER-Konzeption, EDVA ROBOTRON 21

Das Einheitliche System der Elektronischen Rechentechnik (ESER) wurde arbeitsteilig im Rahmen der Zusammenarbeit der sozialistischen Staaten konzipiert. Das ESER beinhaltet eine leistungsmäßig abgestufte Typenreihe von elektronischen Datenverarbeitungsanlagen, welche die Charakteristika der dritten Rechnergeneration aufweisen. Die im Rahmen des ESER entwickelten EDV-Systeme zeichnen sich durch folgende wesentliche Merkmale aus:

- einheitliche Grundkonzeption
- abgestufte Leistungsparameter der Zentraleinheiten
- Anschlußmöglichkeiten der gleichen peripheren Geräte an alle Zentraleinheiten
- Datenkompatibilität
- Verwendung einheitlicher Betriebssysteme (DOS/ES und OS/ES)
- Programmkompatibilität innerhalb eines Betriebssystems.

Tafel 1 gibt einen Überblick über die Modelle des ESER und deren wesentliche Parameter.

Die in der DDR entwickelte elektronische Datenverarbeitungsanlage ROBOTRON 21 stimmt mit den Systemeigenschaften des ESER voll überein. Die im Rahmen des ESER erbrachten gerätetechnischen als auch systemunterlagenseitigen Leistungen sind für die EDVA ROBOTRON 21 nutzbar.

Im allgemeinen wird bei den Datenverarbeitungsanlagen der dritten Rechnergeneration zwischen dem Maschinensystem und dem Betriebssystem unterschieden. So z. B. geht die Konzeption des ESER sowie der EDVA ROBOTRON 21 von dieser Zweiteilung aus.

Unter dem Begriff *Maschinensystem* soll die gerätetechnische Seite des EDV-Systems verstanden werden. Das *Betriebssystem* ist nach *Eilitz* [1] „für eine optimale betriebliche Nutzung der vielfältigen Geräteeinrichtungen die erforderliche programmierte Erweiterung des technischen Systems". Im Bild 1 ist diese Zweiteilung nochmals herausgestellt. Betriebssystem und Maschinensystem bedingen einander. Beide Komponenten bestimmen die Effektivität des EDV-Systems.

Um die größeren Modelle des ESER (Tafel 1) optimal zu nutzen, steht neben dem Betriebssystem DOS/ES ein weiteres Betriebssystem, das OS/ES, zur Verfügung. Für die EDVA R10 (ES 1010) wird ein spezielles Operationssystem (OS/R10) bereitgestellt.

Entsprechend Bild 1 gehören zum Betriebssystem Steuerprogramme, Verwaltungsprogramme (Serviceprogramme) und Arbeitsprogramme.

In der Gruppe der *Steuerprogramme* werden die Programme des Betriebssystems zusammengefaßt, die den eigentlichen Programmablauf über-

Tafel 1. Überblick über die ESER-Modelle

	R10 (ES 1010)	R20 (ES 1020)	R30 (ES 1030)	R40 (ES 1040)	R50 (ES 1050)	R60 (ES 1060)	ROBO-TRON 21
Entwicklung/Produktion	VR Ungarn UdSSR	VR Bulgarien UdSSR	VR Polen UdSSR	DDR	UdSSR	UdSSR	DDR
Betriebssystem	OS/R10	DOS/ES	DOS/ES	OS/ES (DOS/ES)	OS/ES (DOS/ES)	OS/ES (DOS/ES)	DOS/ES
HS-Kapazität [K]Bytes	8...64	64...256	128...512	256...1024	128...1024	256...2048	64
Speicherschutz	Schreiben	Lesen und Schreiben	Lesen und Schreiben	Schreiben	Lesen und Schreiben	Lesen und Schreiben	Schreiben
Multiplexkanal							
Zahl der Subkanäle	4	48...120	128	128...256	192 + 4	192 + 4	128
Übertragungsgeschwindigkeit [K]Bytes/s							
a) Multiplexbetrieb	40...100	12	40	20...25	100...450	100...450	13...20
b) Stoßbetrieb	40...300	100	200...300	180...720	450	450	250...400
Selektorkanäle							
Anzahl	1	2	3	6	6	6	1
Übertragungsgeschwindigkeit [K]Bytes/s	50...250	200	600...800	1300	1300	1300	400
Zahl der anschließbaren GSE	32 Geräte	8	8	10	8	8	8

wachen und steuern. In diese Gruppe gehören das zentrale Steuerprogramm, *Supervisor* genannt, das Programm zur Steuerung des Reihenfolgeablaufs der einzelnen vom Anwender geschriebenen Verarbeitungsprogramme, *Jobsteuerung* genannt, und das Programm *Anfangsprogrammlader*.

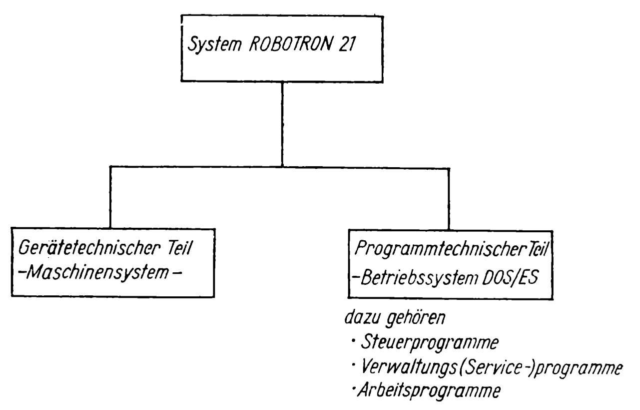


Bild 1. Komponenten eines EDV-Systems

Die Gruppe der *Verwaltungsprogramme* umfaßt Programme zur *Bibliotheksführung* und zum Verbinden von Teilen der vom Anwender geschriebenen Programme. Speziell dafür steht der *Programmverbinder* zur Verfügung.

In der Gruppe der *Arbeitsprogramme* werden *Sprachübersetzer*, *Programme für Datenübertragungen*, *Programme zum Sortieren und Mischen* von Datenfolgen u. a. zusammengefaßt. Dieser Gruppe werden auch die eigentlichen vom Anwender geschriebenen Verarbeitungsprogramme zugeordnet.

Im vorliegenden Band wird hauptsächlich auf die Assemblerprogrammierung eingegangen.

Zur Einleitung ist es günstig, die Gerätetechnik kurz zu behandeln. Dabei kann nicht auf alle Modelle des ESER eingegangen werden. Vielmehr soll im ersten Abschnitt ein Maschinensystem stellvertretend für die Modelle des ESER konkret vorgestellt werden. Dafür bietet sich die EDVA ROBOTRON 21 an. Insbesondere sei der Hinweis gegeben, daß das Betriebssystem der EDVA ROBOTRON 21 mit dem Betriebssystem DOS/ES des ESER identisch ist. Durch ein einheitliches Anschlußbild können alle Ein- und Ausgabegeräte des ESER an die EDVA ROBOTRON 21 angeschlossen werden. Andererseits sind periphere Geräte der EDVA ROBOTRON 21 an Zentraleinheiten von ESER-Modellen anschließbar.

Jedes Modell im ESER sowie die EDVA ROBOTRON 21 verfügt über eine Zentraleinheit, über Kanäle sowie über eine Zahl von Ein- und Ausgabegeräten (E/A-Geräten).

1.1. Zentraleinheit

Die wesentlichen Elemente der Zentraleinheit (ZE) sind der *Hauptspeicher (HS)* und die *zentrale Verarbeitungseinheit (ZVE)*.

1.1.1. *Hauptspeicher*

Die Hauptspeicher (HS) der ESER-Modelle unterscheiden sich u. a. hinsichtlich ihrer *Kapazität*. Die Kapazität wird in K Bytes (1K = 1024) ausgedrückt. Entsprechende Kapazitätsangaben können aus Tafel 1 entnommen werden. Weitere Unterschiede bestehen in der *Zykluszeit* und in der *Zugriffszeit*. Beispielsweise beträgt die Kapazität des HS bei der EDVA ROBOTRON 21 64K Bytes. Die Zykluszeit liegt bei 875 ns, die Zugriffszeit bei 520 ns.
Bei jedem Modell ist das Byte die kleinste adressierbare Einheit. Es besteht aus acht Informationsbits und einem Prüfbit. Das Prüfbit ist aus der Sicht des Anwenders vollkommen uninteressant und soll daher nicht in die weiteren Betrachtungen einbezogen werden.
Jedes Byte wird in zwei *Halbbytes* unterteilt, den *Zonenteil* (Zonenbits) und den *numerischen Teil* (Ziffernbits).

	1 BYTE							
	Zonenteil				Ziffernteil			
	○	○	○	○	○	○	○	○
Bitposition	0	1	2	3	4	5	6	7
Stellenwert	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Mit den acht Informationsbits können 256 verschiedene Zeichen dargestellt werden. Der Arbeitskode (Bild 2) faßt die möglichen Zeichen zusammen.

Ziffernteil 4321	ZL \ SP	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 10	1011 11	1100 12	1101 13	1110 14	1111 15
0000	0					LZ[1]	&	-									0
0001	1							/		a	j			A	J		1
0010	2									b	k	s		B	K	S	2
0011	3									c	l	t		C	L	T	3
0100	4									d	m	u		D	M	U	4
0101	5									e	n	v		E	N	V	5
0110	6									f	o	w		F	O	W	6
0111	7									g	p	x		G	P	X	7
1000	8									h	q	y		H	Q	Y	8
1001	9									i	r	z		I	R	Z	9
1010	10					¢	!	∧	:								
1011	11					.	$	,	#								
1100	12					<	*	%	@								
1101	13					(	)	-	'								
1110	14					+	;	>	=								
1111	15					|	¬	?	"								

Zonenteil: 8765

[1] Leerzeichen

Bild 2. Arbeitskode (EBCDI-Kode) [3] [5] [6]

Datenfelder werden bezüglich des am weitesten links stehenden Bytes des betrachtenden Datenfelds adressiert. Generell erfolgt die Adressierung von links aus fortschreitend.

1.1.2. *Zentrale Verarbeitungseinheit*

Als ZVE sind das Steuerwerk und das Rechenwerk zusammengefaßt.

1.1.2.1. *Steuerwerk*

Im Steuerwerk sind das Zentraleinheitsprogramm, die HS-Schreibsperre, die Zeitgebereinrichtung, das Unterbrechungssystem und andere Einheiten vereinigt.

Durch das *Zentraleinheitsprogramm* wird der Zyklus der Befehlsabarbeitung (Befehlsaufruf, Adreßberechnung, Längenberechnung, Befehlsausführung u. a.) gesteuert.

Zum Schutz der im HS gespeicherten Informationen dient die *HS-Schreibsperre*. Der Anwender kann den HS in 2K Bytes große Blöcke aufteilen. Diesen Blöcken werden Schutzschlüssel zugeordnet, die ein ungewolltes Überschreiben von gespeicherten Informationen während der Programmabarbeitung verhindern.

Die *Zeitgebereinrichtung* ermöglicht unter anderem, nach Ablauf einer vom Anwender eingestellten Zeit eine Unterbrechung des aktuellen Programmlaufs zu erzeugen. Für die Arbeit mit der Zeitgebereinrichtung stehen dem Anwender im Rahmen des Betriebssystems DOS/ES spezielle Makros zur Verfügung.

Die Programmsteuerung wird vom Rechenwerk mit Hilfe des *Unterbrechungssystems* realisiert. Dabei wird davon ausgegangen, daß das laufende Programm aus verschiedenen Gründen unterbrochen werden kann. So z. B. kann der Programmablauf durch einen Maschinenfehler oder durch den Zeitgeberstatus unterbrochen werden. Die möglichen Unterbrechungen werden in folgenden *Unterbrechungsklassen* zusammengefaßt:

1. Externe Unterbrechung
   Darunter fallen: Zeitgeber-Unterbrechung, Betätigen der Unterbrechungstaste am Bedienpult und Unterbrechung durch „externe" Signale.

2. Supervisorruf-Unterbrechung
   Diese Unterbrechung tritt beim Übergang vom eigentlichen Problemprogramm (Anwenderprogramm) in das zentrale Steuerprogramm, den Supervisor, auf.

3. Programmausnahme-Unterbrechung
   Als mögliche Ursachen können auftreten: ungültige Befehlsfolge, Division durch Null, unbekannter Operationskode, Datenfehler, Überlauf bei arithmetischen Operationen u. a.

4. Maschinenfehler-Unterbrechung

5. Eingabe- und Ausgabe-Unterbrechung (E/A-Unterbrechung)
   Eine E/A-Unterbrechung tritt als Endemeldung einer Datenübertragung auf (Abschn. 1.2.).

Generell ist das Auftreten einer Unterbrechung mit einem Wechsel des Arbeitszustands der Zentraleinheit verbunden. Der Ablauf des Problemprogramms wird unterbrochen. Die Steuerung geht an den Supervisor. Dabei kommt dem Supervisor die Aufgabe der Unterbrechungsbehandlung zu. Die Informationen zur Behandlung der aufgetretenen Unterbrechungsbedingung(en) entnimmt der Supervisor dem *laufenden Programmstatuswort (PSW)*. Das PSW enthält Informationen über den aktuellen Zustand des Programms (Statusinformationen). Dieses „laufende" PSW wird

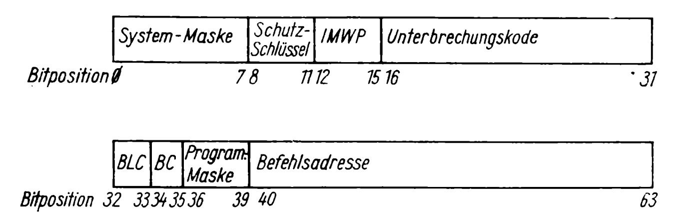


Bild 3. PSW-Aufbau

in einem speziellen Register des Steuerwerks, dem PSW-Register, auf dem aktuellen Stand gehalten. Bild 3 zeigt den Aufbau eines PSW. Die einzelnen Felder sind wie folgt belegt:

Bitposition	Feldname	Bedeutung
0 bis 7	*Systemmaske*	Einzelne Bitpositionen sind dem Selektorkanal (Abschn. 1. 2. 1.), dem Multiplexkanal (Abschn. 1. 2. 2.) sowie der externen Unterbrechung zugeordnet. Es ist möglich, die hier angezeigten Unterbrechungen nicht zu beachten (Maskieren der entsprechenden Bitposition durch Befehl „Setzen Systemmaske").
8 bis 11	*Schutzschlüssel*	Mit der Vergabe von Schutzschlüsseln wird ein unbefugtes Überschreiben der im HS gespeicherten Informationen verhindert.
12 bis 15	*IMWP*	In diesem Feld werden — Kodeauswahlbit (I) — Maschinenfehler-Maskenbit (M) — Wartezustandsbit (W) — Problemzustandsbit (P) angezeigt.

16 bis 31	*Unterbrechungskode*	In diesen Bitpositionen werden die genannten programmbedingten Unterbrechungen angezeigt. Dabei kann die Behandlung wahlweise erfolgen. Das Programm kann abgebrochen werden (mit oder ohne Ausdrucken des Speicherinhalts für Fehleranalyse); es kann aber auch eine spezielle Fehlerroutine des Anwenders zur Fehlerbehandlung aktiviert werden (Fehlermaßnahme).
32 und 33	*Befehlslängenkode*	
34 und 35	*Bedingungskode (BC)*	Die Bits 34 und 35 werden bei der Abarbeitung bestimmter Operationen automatisch belegt (es sind die vier Belegungen 00, 01, 10 und 11 möglich). Dieses Belegen der Bitpositionen wird als „Stellen des Bedingungskodes“ bezeichnet. Im Programm kann auf die Stellung des Bedingungskodes Bezug genommen werden (Abschnitt 7.).
36 bis 39	*Programmaske*	Es ist vom Programm aus möglich, einige auftretende Programmausnahme-Unterbrechungen (Festkomma- und Dezimalüberlauf, Exponentenunterschreitung, Mantisse gleich Null) nicht zu beachten. Die entsprechenden Bitpositionen sind dann zu „maskieren“ (Befehl „Setzen Programmmaske“).
40 bis 63	*Befehlsadresse*	In diesem PSW-Teil wird die Adresse des nächsten abzuarbeitenden Befehls gespeichert (Befehlszähler).

Jede Unterbrechungsklasse hat ihre eigenen PSW – ein altes (APSW) und ein neues (NPSW) –, die an fest zugeordneten HS-Plätzen gespeichert sind. In Abhängigkeit von einer aufgetretenen Unterbrechung erfolgt ein *PSW-Austausch.* Dabei wird das laufende (steuernde) PSW (SPSW) auf dem Speicherplatz des zugehörigen APSW gespeichert. Das entsprechende NPSW wird in das PSW-Register als SPSW übernommen. Durch diesen PSW-Austausch wird das laufende Programm

unterbrochen. Die Fortsetzung der Arbeit erfolgt mit der Adresse, die im NPSW (jetzt SPSW) gespeichert ist. Das unterbrochene Programm kann fortgesetzt werden, indem das abgespeicherte PSW geladen wird, d. h. zum SPSW (durch Befehl „Laden PSW“) gemacht wird.

1.1.2.2. *Rechenwerk*

Mit Hilfe des Rechenwerks werden arithmetische und logische Operationen ausgeführt. Das Rechenwerk kann Binärzahlen, Dezimalzahlen und logische Daten verarbeiten. Dazu verwendet das Rechenwerk 16 allgemeine Register (Adressen 0 bis 15) mit je 32 Bits Speicherkapazität. Für die Realisierung der Gleitkommaarithmetik stehen dem Anwender zusätzlich 4 Gleitkommaregister (Adressen 0, 2, 4, 6) zur Verfügung (Bild 4).

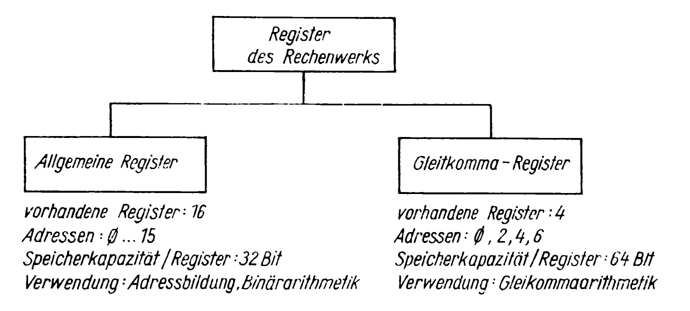


Bild 4. Register der ESER-Modelle und der EDVA ROBOTRON 21

1.2. Ein- und Ausgabesystem

Zum Ein- und Ausgabesystem (E/A-System) zählen die *Kanäle, Gerätesteuereinheiten (GSE)* und die eigentlichen *E/A-Geräte.* Durch Anwendung des Hierarchieprinzips bei der Konzeption des E/A-Systems ist es möglich, daß E/A-Operationen parallel zu internen Operationen ablaufen können. Die E/A-Steuerung ist im Bild 5 dargestellt. Die ZE „stößt“ die E/A-Ope-

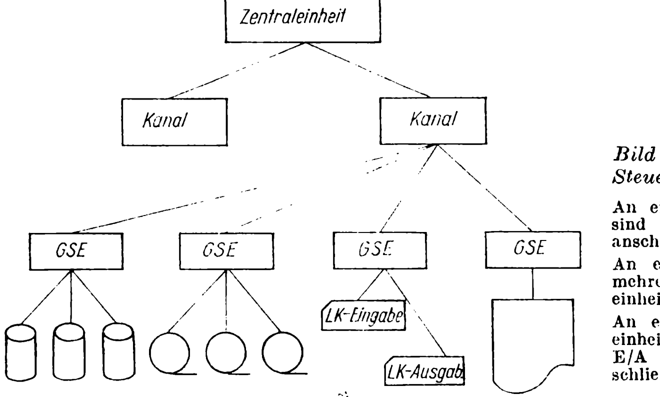


Bild 5. E/A-Steuerprinzip [4]

An eine Zentraleinheit sind mehrere Kanäle anschließbar

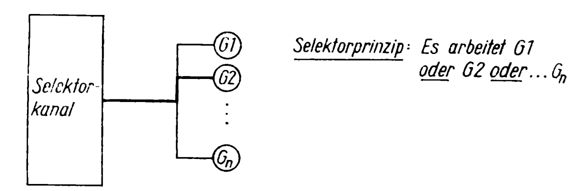
An einem Kanal sind mehrere Gerätesteuereinheiten anschließbar

An eine Gerätesteuereinheit sind mehrere E/A Einheiten anschließbar

ration an und delegiert die E/A-Anforderungen an den Kanal. Der Kanal übernimmt die weitere E/A-Steuerung und meldet der ZE das Ende der E/A-Operation durch eine E/A-Unterbrechung. Der Kanal hat seinen eigenen Befehlssatz. Die Befehle an den Kanal heißen *Kanalkommandoworte (CCW)*. Durch Abarbeiten des Kanalprogramms, einer Folge von CCW, realisiert der Kanal seine Aufgaben.
Die Modelle des ESER und die EDVA ROBOTRON 21 sind mit je einem Multiplexkanal und einer unterschiedlichen Anzahl von Selektorkanälen ausgerüstet. Entsprechende Angaben können Tafel 1 entnommen werden.

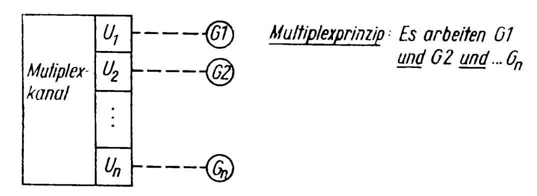
1.2.1. *Selektorkanal*

Der Selektorkanal ist für den Anschluß der schnellen E/A-Geräte (Magnetplatte, -band) vorgesehen. Auf Grund der hohen Übertragungsfrequenz bedient der Kanal im gleichen Zeitraum nur eine GSE. Diese GSE bedient ihrerseits im betrachteten Zeitraum nur ein E/A-Gerät. Der Datenaustausch erfolgt satzweise. Diese Übertragungsart wird als *Stoßbetrieb* bezeichnet.



1.2.2. *Multiplexkanal*

An den Multiplexkanal werden normalerweise „langsame" E/A-Geräte (z. B. Drucker, Lochbandleser, -stanzer) angeschlossen. Die relativ langsam arbeitenden E/A-Geräte gestatten es, daß der Kanal mehrere GSE gleichzeitig bedient. Deshalb ist der Multiplexkanal in mehrere *Unterkanäle* (auch Subkanäle genannt) unterteilt. Jeder Unterkanal bedient eine GSE. Dabei erfolgt der Datenaustausch byteweise.



Werden schnelle E/A-Geräte an den Multiplexkanal angeschlossen, erzwingen sie Stoßbetrieb.

1.2.3. *Gerätesteuereinheiten*

Die GSE ist das Bindeglied zwischen Kanal und E/A-Gerät. Sie kann als selbständige Baueinheit ausgeführt oder im E/A-Gerät mit untergebracht sein. An eine selbständige GSE (z. B. bei Magnetband oder Magnetplatte) können maximal 8 E/A-Geräte angeschlossen werden.

1.2.4. *E/A-Einheiten*

Bei den Modellen des ESER und der EDVA ROBOTRON 21 ist zwischen den Kanälen und den Gerätesteuereinheiten ein einheitliches Anschlußbild geschaffen worden. Dadurch ist eine universelle Koppelbarkeit der ZE mit den E/A-Geräten über die Kanäle möglich (Bild 6). Dieses einheitliche Standardanschlußbild (auch *S*tandard*i*nter*f*ace genannt) soll kurz mit „SIF ESER" bezeichnet werden. Der prinzipielle Zusammenhang zwischen den Kanälen, dem SIF ESER und den E/A-Geräten ist im Bild 6 am Beispiel der EDVA ROBOTRON 21 dargestellt. Bei dieser Darstellung wurde die Form eines Blockschaltbilds gewählt.
Im folgenden sollen die E/A-Geräte in Kurzform charakterisiert werden. Dabei werden die ESER-Bezeichnungen mit angegeben.

1.2.4.1. *Abfrageeinheit 7073*

Funktion. Vorwiegender Einsatz für Bedienung des Systems als Protokollschreibmaschine sowie zur Kommunikation (Zweiwegeverständigung) zwischen Bediener und System; die Abfrageeinheit besteht aus GSE und dem Schreibwerk.

Daten

Schreibmaschine Soemtron 529-221
Geschwindigkeit (Ausgabe) 10 Zeichen/s

1.2.4.2. *Paralleldrucker 7031/7035*

Funktion. Ausgabegerät zum Auflisten von Informationen. Der Drucker steht in zwei Modellen zur Verfügung. Er besteht aus dem Druckwerk und der GSE.

Daten

	Modell 7031	Modell 7035
Druckgeschwindigkeit (Zeilen/min)	900	600
Druckstellen/Zeile	156	120
Papierbahnen	1 oder 2	1
Druckprinzip	fliegender Druck mit Farbtuch	
Formularvorschub	durch Lochband oder durch Kommandos	

1.2.4.3. *Lochkartenleser 6012*

Funktion. Systemeingabegerät für die Arbeit des Betriebssystems und Eingabegerät für Lochkartendateien.

Daten

Lesegeschwindigkeit	500 Karten/min
Leseprinzip	seriell, ab Lochkartenspalte 1 beginnend
Leseverfahren	fotoelektrisch
Kartenart	80spaltige Lochkarten

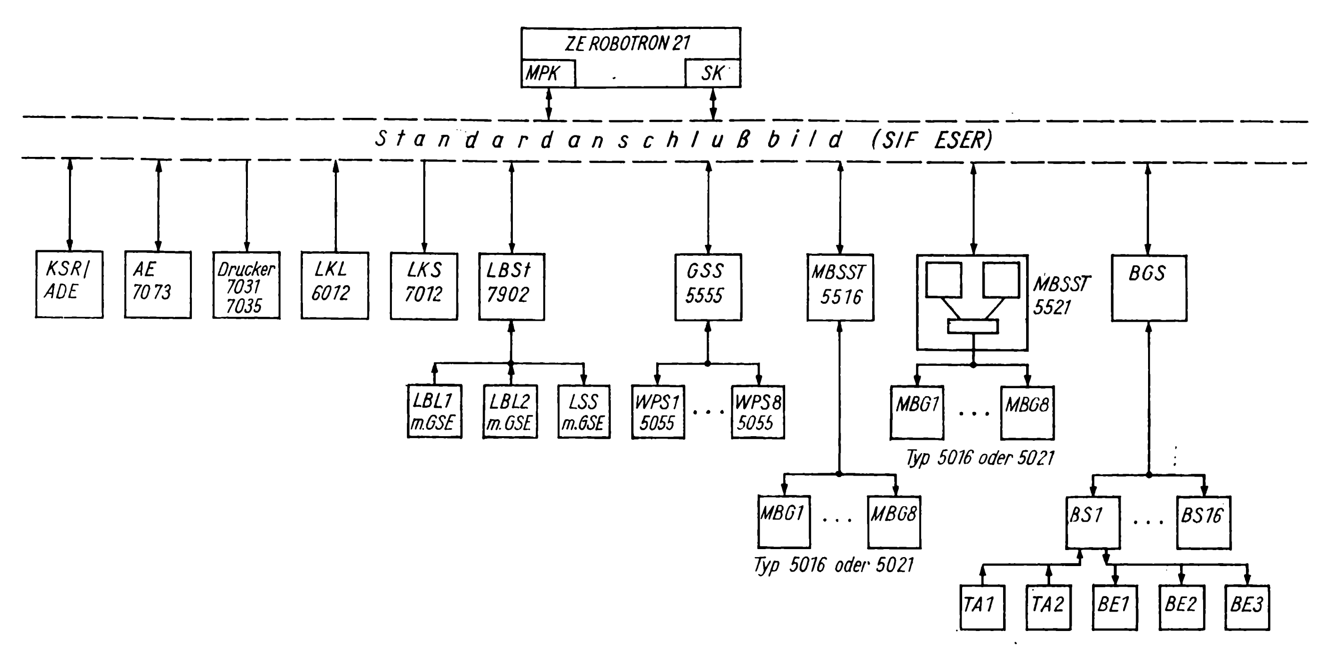


Bild 6. Blockschaltbild der EDVA ROBOTRON 21

AE	Abfrageeinheit	*MBSST*	Magnetbandspeicher-Steuergerät
LKL	Lochkartenleser	*MBG*	Magnetbandgerät
LKS	Lochkartenstanzer	*BGS*	Bildschirmgruppen-Steuergerät
LBSt	Lochbandstation	*BS*	Bildschirmsteuergerät
LBL	Lochbandleser	*TA*	Tastatur
LSS	Lochbandstanzer	*BE*	Bildschirmeinheit
GSS	Großraumspeicher-Steuereinheit	*KSR/ADE*	Steuereinheit für Datenerfassung und -ausgabe
WPS	Wechselplattenspeicher		

1.2.4.4. *Lochkartenstanzer 7012*

Funktion. Stanzgerät für Lochkarten-Ausgabedateien.

Daten

Stanzprinzip	seriell, ab Lochkartenspalte 1 beginnend
Stanzgeschwindigkeit	abhängig von zu stanzender Informationsmenge; bei den ersten 20 Spalten/LK: 10 000 LK/h bei 80 Spalten/LK: 5 000 LK/h
Kartenart	80spaltige Lochkarten

1.2.4.5. *Lochbandstation 7902*

Funktion. E/A-Gerät für Datenträger Lochband. Maximal anschließbar: zwei Lochbandleser und ein Lochbandstanzer.
Jedes Gerät der Station hat eine eigene GSE und belegt einen Subkanal.

Daten

1. Leser	Lesegeschwindigkeit	1000 Zeichen/s
	Spuranzahl	5...8
	Kode	a) beliebig b) nach TGL 23 207
2. Stanzer	Stanzgeschwindigkeit	100 Zeichen/s
	Spuranzahl	5...8
	Kode	vgl. Lochbandleser

1.2.4.6. *Wechselplattenspeicher 5055*

Funktion. Externer Speicher mit wahlfreiem Zugriff zur Speicherung großer Datenmengen.
An einen Kanal sind maximal acht Großraumspeichersteuereinheiten (GSS) des Typs 5555 anschließbar. An eine GSS (GSE für Wechselplattenspeicher) können maximal acht Speichereinheiten angeschlossen werden. Jede Speichereinheit kann einen Plattenstapel betreiben. Dieser Plattenstapel ist auswechselbar (Wechselplattenspeicher). Ein Plattenstapel besteht aus sechs übereinander angebrachten Magnetplatten, wobei von den zwölf verfügbaren Plattenoberflächen zehn für Informationsspeicherung benutzt werden.

Daten

Zylinder/Plattenstapel	200 + 3 (Ersatz)
Spuren/Zylinder	10
Speicherkapazität/Spur	3625 Bytes
Speicherkapazität/Plattenstapel	$7{,}25 \cdot 10^6$ Bytes
anschließbare Plattenstapel/GSS	maximal 8
Positionierzeit (minimal)	30 ms
(maximal)	165 ms

1.2.4.7. *Magnetbandspeichergeräte*

Funktion. Externer Speicher für sequentiell organisierte Dateien.
Es können verschiedene Typen von Steuereinheiten und Bandgeräten angeschlossen werden:

a) Magnetbandspeichersteuergerät (MBSST) vom Typ 5516 für den Anschluß von maximal acht Magnetbandgeräten 5016 oder 5021

b) Magnetbandspeichersteuergerät Typ 5521 mit 2-Kanal-Anschluß für den Anschluß von maximal acht Magnetbandgeräten 5016 oder 5021.

Anmerkung. Beim MBSST Typ 5516 kann nur eins der maximal acht anschließbaren MB-Geräte in Datenaustausch mit dem Kanal treten. MBSST Typ 5521 ermöglicht durch den „2-Kanal-Anschluß" eine Datenübertragung zweier MB-Geräte gleichzeitig über die gleiche GSE.

Daten

	Typ 5016	Typ 5021
Bandgeschwindigkeit	1,5 m/s	3 m/s
Übertragungsgeschwindigkeit bei einer Aufzeichnungsdichte von		
8 Bit/mm	12 kHz	24 kHz
32 Bit/mm	48 kHz	96 kHz
Spuranzahl	9	
Bandlänge	750 m	
Bandbreite	1/2 Zoll	
Blocklücke	15,24 mm	
Rückwärtslesen	möglich	
anschließbare Magnetbandspeichergeräte/GSE	maximal 8	

1.2.4.8. *Steuereinheit für Datenerfassung und -ausgabe*

Funktion. Bindeglied zwischen ZE und den verschiedenen Datenerfassungs- und -ausgabestellen. Rechnerseitig hat die Steuereinheit das Standardanschlußbild (SIF ESER).
Geräteseitig kann an die Steuereinheit folgendes Gerätespektrum angeschlossen werden:

- dezentrale Abfrageeinheit (DZA)
- Bildschirmsteuergerät (BS)
- Datenfernübertragungseinrichtung (DFÜ)
- Datennahübertragungseinrichtung (DNÜ).

1.2.4.9. *Bildschirmsystem*

Funktion. Optischer Informationsaustausch zwischen Nutzer und System. Das Bildschirmsystem besteht aus dem Bildschirmgruppensteuergerät, den Bildschirmsteuergeräten und den Bildschirmarbeitsplätzen mit Bildschirmeinheit, Tastatur und Lichtstift. Das Bildschirmgruppensteuergerät hat rechnerseitig das Standardanschlußbild SIF ESER.

Daten

Zeichenvorrat	64 Zeichen
Zeichenzahl/Zeile	64
Zeilenzahl	16

Bildgröße	210 × 115 mm
Informationslänge	variabel, maximal 1024 Zeichen je E/A-Operation
Entfernung zwischen Bildschirmgruppensteuergerät und Bildschirmsteuergerät	maximal 1000 m
Zahl der maximal anschließbaren Bildschirmsteuergeräte an ein Bildschirmgruppensteuergerät	16

2. Zeichendarstellung, Daten- und Befehlsformate

Im Abschn. 1.1.1. wurde das Byte als kleinste adressierbare Speichereinheit genannt. Je Byte können entweder

2 Ziffern (je Halbbyte eine Ziffer) oder
1 Ziffer und 1 Vorzeichen oder
1 Alphazeichen (Buchstabe oder Sonderzeichen)

dargestellt werden. Die Kodierung der Zeichen kann intern im erweiterten binärkodierten Dezimalkode (EBCDI-Kode) oder im erweiterten 7-Bit-Kode nach TGL 23207

erfolgen. In den weiteren Betrachtungen wird auf den *EBCDI-Kode* orientiert (s. Arbeitskode, Abschn. 1.1.1.).

Prinzipiell lassen sich die in der EDVA zu verarbeitenden Daten in

alphanumerische Daten und
rein numerische Daten

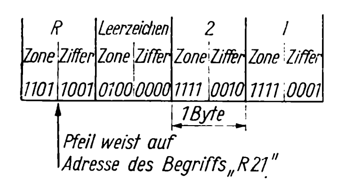
gliedern.

2.1. Speicherung alphanumerischer Daten

Unter *alphanumerischen Zeichen* sollen Ziffern, Buchstaben und Sonderzeichen verstanden werden.
Die Kodierung jedes Zeichens ist aus dem Arbeitskode (Abschn. 1.1.1.) ersichtlich.

Beispiel

Es soll die Zeichenfolge R 21 intern kodiert werden.



Bemerkung. Zwischen dem Buchstaben R und der Zahl 21 soll ein Leerzeichen (Wortzwischenraum) stehen.

2.2. Speicherung dezimaler Zahlenwerte

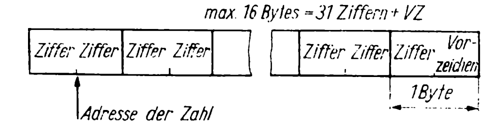
Bei der Speicherung dezimaler Zahlenwerte ist zwischen der Speicherung im *ungepackten Format* und der im *gepackten Format* zu unterscheiden.

2.2.1. *Gepackte Speicherungsform*

Um die Ziffern des Dezimalsystems (0 bis 9) darstellen zu können, sind vier Bitpositionen notwendig. Es bietet sich somit an, zwei Ziffern in einem Byte zu speichern (zu packen). Bei dieser Form wird von der Darstellung gepackter Zahlen gesprochen. Je Halbbyte wird eine Ziffer gespeichert. Dabei entsprechen den Ziffern folgende Bitkombinationen:

Ziffer	Bitkombination				
	2^3	2^2	2^1	2^0	(Stellenwert)
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	usw.
.					
.					
.					
9	1	0	0	1	

Das Vorzeichen der Zahl wird im rechten Halbbyte des am weitesten rechts stehenden Bytes gespeichert.

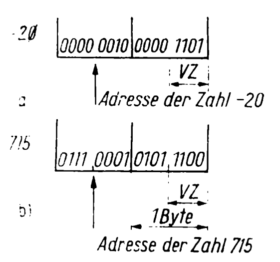


Die Vorzeichendarstellung ist wie folgt festgelegt:

Vorzeichen	Bitkombinationen
+	1100 oder
	1111
−	1101

Beispiel

Es sind die Zahlen a) −20 und b) 715 im gepackten Format intern darzustellen.

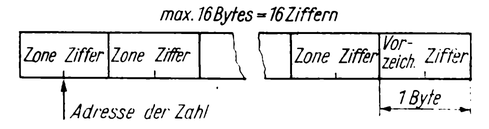


Bemerkung zu Punkt a. Es wird immer auf volle Bytes ergänzt (Auffüllen des linken Halbbytes des linken Bytes mit Bitkombination 0000). Die

gepackte Darstellung ist für das Anwenden der Dezimalarithmetik (Abschn. 6.) notwendig.

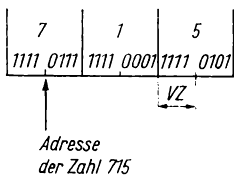
2.2.2. *Ungepackte Speicherungsform*

Die ungepackte Speicherungsform (auch als erweiterte oder gezonte Form bezeichnet) ist für die Ein- und Ausgabe von Daten notwendig. Bei ihr wird in einem Byte eine Ziffer oder eine Ziffer und ein Vorzeichen gespeichert. Dabei wird das Vorzeichen der Zahl in dem linken Halbbyte des am weitesten rechts stehenden Bytes verschlüsselt.



Beispiel

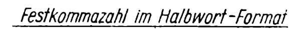
Die Zahl 715 soll in ungepacktem Format intern dargestellt werden.

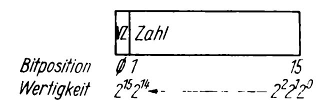


2.3. Speicherung dualer Zahlenwerte

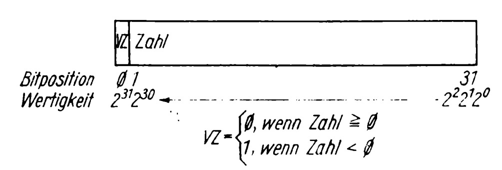
Die Speicherung dualer Zahlen (Binärzahlen) ist bei der Anwendung der *Binärarithmetik* (Abschn. 8.) und der *Gleitkommaarithmetik* notwendig. Auf die Gleitkommaarithmetik wird in den weiteren Betrachtungen nicht eingegangen.

Die Binärarithmetik verarbeitet Binärzahlen im *Halbwortformat* (2 Bytes) und *Wortformat* (4 Bytes). Die Zahlen werden als vorzeichenbehaftete ganze Zahlen aufgefaßt. Die Belegung der Bitposition 0 wird als Vorzeichen der Zahl gedeutet.





Festkommazahl im Wortformat



Die binären Festkommazahlen können durch Umwandlung aus Dezimalzahlen entstehen (s. Datenkonvertierung, Abschn. 8.1.) oder sie werden durch Definitionsbefehle (Abschn. 4.3.4.) in das Programm eingefügt.

Bei der Speicherung binärer Festkommazahlen sind die Adressen der HS-Plätze nicht willkürlich wählbar. So muß eine Festkommazahl im Halbwortformat an einer durch 2 teilbaren Adresse beginnen, eine Festkommazahl im Wortformat an einer durch 4 teilbaren Adresse. Für bestimmte Operationen wird ein Doppelwortformat (8 Bytes) verlangt, die entsprechende Speicheradresse muß durch 8 teilbar sein. In diesem Zusammenhang wird von „integraler“ Grenze gesprochen. Damit ist gemeint, daß die erforderliche Speicheradresse ein Vielfaches der Länge des entsprechenden Datenformats in Bytes beträgt. In der Binärarithmetik (Abschn. 8.) wird auf diesen Sachverhalt ständig verwiesen.

2.4. Darstellung in hexadezimaler Schreibweise

In einem Halbbyte (4 bit) können $2^4 = 16$ verschiedene Bitkombinationen speichertechnisch realisiert werden. Dabei werden den Ziffern 0 bis 9 die Bitkombinationen 0000 bis 1001 zugeordnet. Es besteht die Forderung, alle 16 Möglichkeiten durch ein einstelliges Zeichen auszudrücken. Dadurch wird es möglich, jede beliebige Bitkombination im HS durch ein *hexadezimal dargestelltes Zeichen* zu beschreiben und z. B. über Drucker auszugeben. Dabei wurde die sog. *hexadezimale Darstellung* gewählt. Diese Darstellungsform kann als Zahlensystem mit der Basis 16 und den hexadezimalen Ziffern 0 bis 9 und A bis F aufgefaßt werden. Im einzelnen wurde folgende Zuordnung getroffen:

Dezimale Darstellung	Duale Darstellung	Hexadezimale Darstellung
0	0000	0
1	0001	1
.	.	.
.	.	
	.	
	.	
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Um die Hexadezimalziffern 0 bis F nicht mit den Zeichen 0 bis F (s. Arbeitskode, Abschn. 1.1.1.) zu verwechseln, werden die Hexadezimalzahlen durch ein vorangestelltes X mit der darauffolgenden in Hochkomma Apostroph) eingeschlossenen Hexadezimalzahl gekennzeichnet.

Beispiel

Es soll die Dezimalzahl 26 als Hexadezimalzahl dargestellt werden.

Dezimalzahl 26 dual dargestellt:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Stellenwert
0	0	0	1	1	0	1	0	

Also: $26_{10} \mathrel{\hat{=}} 11010_2$

Anwendung des Hexadezimalsystems:

16^1	16^0	Stellenwert
0 0 0 1	1 0 1 0	

Also: $26_{10} \mathrel{\hat{=}} 11010_2 \mathrel{\hat{=}} 1A_{16}$, denn $1 \cdot 16^1 + 10 \cdot 16^0 = 26$

Schreibweise: X'1A'

***Bemerkung*:** Bei der hexadezimalen Darstellung geht es nicht um eine spezielle Speicherungsform, sondern um eine Darstellungsform beliebiger Bitkombinationen innerhalb eines Halbbytes. Eine Übersicht über mögliche Speicherungsformen und Datenformate wird im folgenden Abschnitt gegeben.

Übung

Ü. 2.1. Es sollen folgende Dezimalzahlen intern dargestellt werden:
- a) gepackt —10 und 400
- b) ungepackt 12

Ü. 2.2. Folgende Zahlen sind hexadezimal darzustellen:
- a) 21
- b) 16
- c) 37

(Lösungen oder Lösungshinweise am Ende des Bandes)

2.5. Übersicht über Datenformate und Speicherungsformen

Im Bild 7 sind die möglichen Datenformate zusammengestellt. Datenformate, Adressierung der Daten und ihre Verarbeitung stehen im engen Zusammenhang. Im Bild 8 ist dieser Zusammenhang in Form einer Übersicht aufgezeigt.

2.6. Befehlsformate

Die Modelle des ESER sowie die EDVA ROBOTRON 21 arbeiten mit Maschinenbefehlen unterschiedlicher Struktur. Diese Befehle unterscheiden sich im Aufbau und in ihrer Wirkungsweise. Grundsätzlich lassen sich diese Befehle in fünf Gruppen einteilen. Die Befehlsgruppen unterscheiden sich voneinander, wobei die Struktur der Befehle innerhalb jeder Gruppe gleich ist. Jeder Befehl hat ein bestimmtes Format, sein Befehlsformat. Somit kann, entsprechend der Einteilung der Befehle in fünf Gruppen, von fünf Befehlsformaten gesprochen werden.

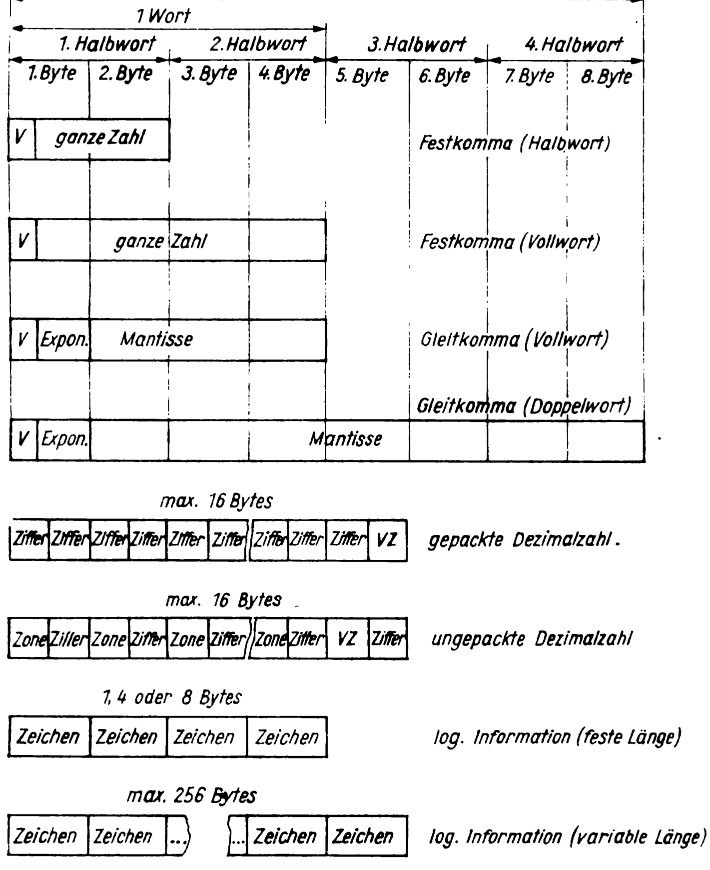


Bild 7. Datenformate [4]

Jedes einzelne Befehlsformat stellt bestimmte Anforderungen bezüglich Struktur und Adressierung an die zu verarbeitenden Daten (Operanden). So z. B. können durch einen Befehl ein Operand, durch einen anderen Befehl eines anderen Befehlsformats mehrere Operanden verarbeitet werden, oder ein Befehl setzt Operanden im gepackten Format voraus, ein anderer Befehl (eines anderen Formats) erfordert Operanden in Form von Binärzahlen.

Trotz des unterschiedlichen Formats der Befehle kann von einem grundsätzlichen Befehlsaufbau gesprochen werden. Dieser *grundsätzliche Befehlsaufbau* ist wie folgt darstellbar:

WAS	WIEVIEL	WOHIN	WOHER

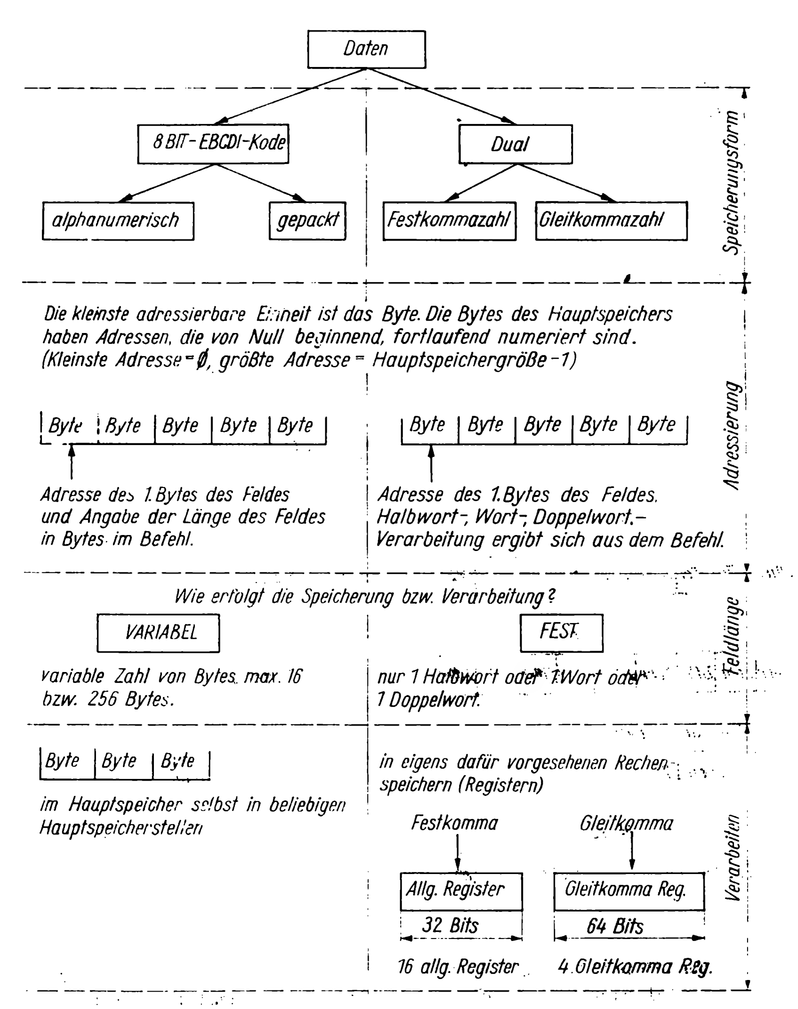


Bild 8. Übersicht über Speicherungsform, Adressierung und Verarbeitung von Daten [4].

Ein Befehl sagt somit prinzipiell aus:

WAS	ist zu tun (Operationsschlüssel)?
WIEVIEL	Bytes sind zu verarbeiten (Feldlänge)?
WOHIN	kommen die Daten (Adresse des ersten Feldes)?
WOHER	kommen die Daten (Adresse des zweiten Feldes)?

WAS — Der Operationsschlüssel gibt an:
welche Operation ausgeführt werden soll
ob Daten fester oder variabler Länge verarbeitet werden sollen
ob die Daten in dezimaler oder binärer Form vorliegen
ob die Daten im Register oder im Hauptspeicher stehen
wie lang der Befehl ist.

WIEVIEL — Die Feldlänge gibt an:
wieviel Bytes durch den Befehl verarbeitet werden. Bei Befehlen, die sich auf Felder fester Länge beziehen, entfällt diese Angabe.

WOHIN — Die Adresse des ersten Feldes kennzeichnet die Speicherstellen, auf denen
vor Ausführung des Befehls der erste Operand steht
nach Ausführung des Befehls das Ergebnis zu finden ist.

WOHER — Die Adresse des zweiten Feldes kennzeichnet die Speicherstellen, auf denen der zweite Operand beginnt bzw. steht. Die beiden Operanden können im Hauptspeicher oder in Registern stehen. Ist ein Wert in einem Register gespeichert, so ist seine Adresse die Nummer des Registers (0 bis 15).

Die allgemeinen Aussagen sollen an einem Beispiel diskutiert werden.

Beispiel.

Der Inhalt des Feldes mit der symbolischen Adresse BER1 soll im Feld BER2 zwischengespeichert werden. Dafür wird ein Transportbefehl mit dem Operationskode MVC (Abschn. 5.1.) verwendet.

```
         MVC   BER2,BER1
```

Dieser Befehl sagt aus

1. WAS für eine Operation auszuführen ist: Der Operationskode MVC (move characters) steht symbolisch für eine Transportoperation;
2. WIEVIEL Bytes zu transportieren sind: Diese Angabe ergibt sich aus der vereinbarten Länge für das Feld BER2. Beispielsweise sei dem Feld BER2 eine Länge von 8 Bytes zugeordnet. Dann werden 8 Bytes in die Transportoperation einbezogen. Ist die Feldlänge für BER2 mit 6 Bytes vereinbart, werden nur 6 Bytes übertragen usw.;
3. WOHIN die Bytes transportiert werden sollen: Diese Zieladresse ist im Beispiel die symbolische Adresse BER2;
4. WOHER die zu transportierenden Bytes kommen: Diese Herkunftsadresse ist im Beispiel durch die symbolische Adresse BER1 gegeben.

Die genannte Befehlsgrundstruktur ist prinzipiell bei jedem Befehlsformat gegeben. Die einzelnen Befehlsformate sollen im folgenden näher erläutert werden (Bild 9).

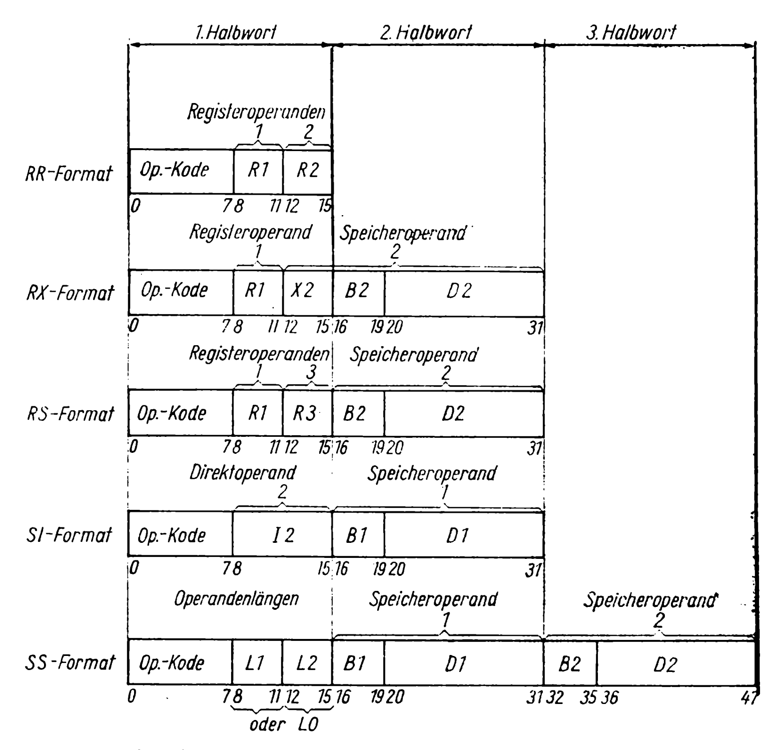


Bild 9. Befehlsformate [4]

RR-Format (Register — Register)

Beide Operanden stehen in den adressierten allgemeinen Registern (Adressen 0 bis 15).

RX-Format (Register — Speicher)

Der erste Operand befindet sich in einem allgemeinen Register, der zweite Operand im HS. Seine HS-Adresse kann durch den Inhalt eines Registers modifiziert werden (Abschn. 9.).

RS-Format (Register—Speicher)

Befehle im RS-Format enthalten zwei Registeradressen und eine Hauptspeicheradresse. Sie werden verwendet bei Mehrfachlade- und Mehrfachspeicheroperationen (Abschn. 8.2.1.) sowie beim Aufbau zyklischer Programme (Abschn. 9.3.).

SI-Format (Speicher — Direktwert)

Ein Operand befindet sich im HS, der zweite ist als „Direktoperand“ im Befehl enthalten.

SS-Format (Speicher—Speicher)

Beide Operanden befinden sich im HS. Beim SS-Format ist zwischen
arithmetischem SS-Format (Operandenlängen L1, L2) und
logischem SS-Format (Operandenlänge L0)
zu unterscheiden.

Befehlsformate und zugehörige Operandenlängen können wie folgt zusammengefaßt werden:

Befehlsformat	Zugehörige Operanden-länge	Entsprechendes Assemblerbefehlsformat
RR	fest: 1 Wort	OpK[1]) R1,R2
RX	je nach Art der Operation fest: 1 Wort oder 1 Halbwort	OpK R1,D2(X2,B2)
RS	je nach Art der Operation fest: 1 oder mehrere Worte	OpK R1,R3,D2(B2)
SI	fest: 1 Byte	OpK D1(B1),I2
SS (logisch)	variabel: 1 bis 256 Bytes	OpK D1(L0,B1),D2(B2)
SS (arithmetisch)	variabel: 1 bis 16 Bytes	OpK D1(L1,B1),D2(L2, B2)

Die Befehle werden im Assemblerformat geschrieben. Die Umwandlung in das interne Befehlsformat (Bild 9) erfolgt durch das Übersetzungsprogramm, den Assembler (Abschn. 4.).

2.7. Bildung von Operandenadressen

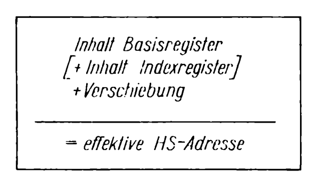
Die Operanden sollen wie folgt klassifiziert werden:

Registeroperanden
Direktoperanden
HS-Operanden.

Für die Adressierung von HS-Operanden stehen 16 Bitpositionen zur Verfügung. Damit könnten 2^{16} Bytes = 2^6 K Bytes = 64 K Bytes adressiert werden. Das wäre für das System ROBOTRON 21 ausreichend. Bei der Aufwärtskompatibilität zu größeren Systemen der ESER-Reihe ist diese Adressierungsart nicht geeignet. Deshalb wird mit einer *indirekten Adressierungstechnik* gearbeitet.
In den Adreßfeldern der Befehle, die sich auf HS-Operanden beziehen, ist eine *Basisregisternummer (B)* und eine *Verschiebung (D)* angegeben.

[1]) OpK = Operationskode

Für die Basisregisternummer stehen 4 Bitpositionen zur Verfügung. Den adressierten Basisregistern entsprechen die allgemeinen Register 0 bis 15. Im Basisregister selbst wird in 24 Bitpositionen eine Basisadresse gespeichert. Diese Basisadresse kann mit einem Anfangspunkt der Programmadressierung (z. B. Programmbeginn bei 20 000) verglichen werden. Relativ bezüglich des Anfangspunktes (Basisadresse) ergibt sich für jedes im Programm zu adressierende Element (Feld, Bereich) eine Verschiebung (D). Sie wird in Bytes angegeben und drückt den Abstand (Distanz) des jeweilig adressierten Elements von der Basisadresse aus. Für diese Verschiebung stehen 12 Bitpositionen zur Verfügung. Dadurch kann mit einem Basisregister ein Programmbereich von $2^{12} = 4096$ Bytes „überdeckt" werden.
Die eigentliche Adreßrechnung erfolgt zum *Zeitpunkt der Programmausführung* nach folgender Regel:



Bemerkung: Zur Bildung der effektiven HS-Adresse wird *nur* bei den Befehlen im RX-Format der Inhalt eines Indexregisters mit verwendet. Ansonsten wird die effektive HS-Adresse aus dem Inhalt eines Basisregisters plus einer Verschiebung gebildet.

Beispiel

Ein Befehl im SI-Format soll diesen Sachverhalt verdeutlichen (Bild 10).

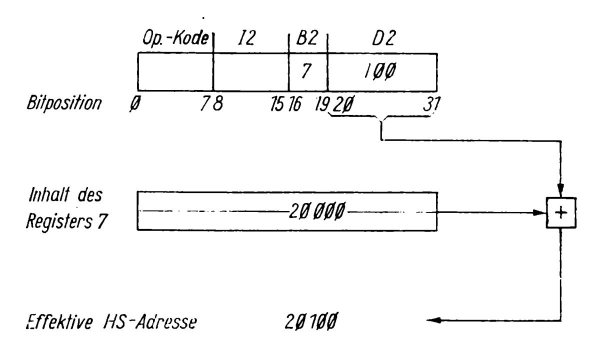
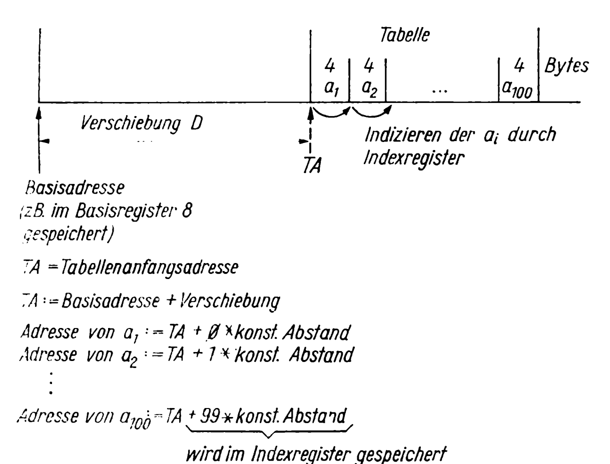


Bild 10. Berechnung von Operandenadressen

Bemerkung: Bei der erläuterten Adressierungstechnik können auf Grund der zur Adressierung verwendeten 24 Bitpositionen 2^{24} Adressen gebildet werden. Damit könnte prinzipiell ein Hauptspeicher mit 2^{14} K (= 16 384K) Bytes Speicherkapazität überstrichen werden.
In den bisherigen Darlegungen wurden die Begriffe ***Basisregister***, ***Indexregister*** und ***allgemeines Register*** verwendet. Diese Begriffe sind inhaltlich

voneinander abzugrenzen. Zusammenfassend soll diese Abgrenzung dargestellt werden.
Entsprechend Bild 4 verfügen die Modelle des ESER sowie die EDVA ROBOTRON 21 über 16 *allgemeine Register* mit den Adressen 0 bis 15. Diese 16 allgemeinen Register werden für verschiedene Zwecke verwendet. Einerseits werden einige aus der Menge der allgemeinen Register zu arithmetischen Operationen bei Anwendung der Binärarithmetik verwendet, andererseits werden allgemeine Register zur Adressierung der Operanden benötigt.
In jedem Programm muß *mindestens ein* allgemeines Register als *Basisregister* fungieren. In diesem Register wird eine Basisadresse (Anfangspunkt des Programms) gespeichert. Dieses (oder diese) Register kann (können) im Programm für keinen anderen Zweck verwendet werden. Die Zuordnung, welches Register aus der Menge der allgemeinen Register als Basisregister verwendet werden soll, erfolgt am Anfang des Programms mit Hilfe der Assembleranweisung USING (Abschn. 4.3.3.).
Mit einem Basisregister kann ein HS-Bereich von 4096 Bytes überdeckt (adressiert) werden. *Innerhalb* dieses HS-Bereichs sollen z. B. Daten adressiert werden, die tabellarisch gespeichert sind. Dabei sollen diese Daten konstanten Aufbau und konstanten Abstand voneinander haben. Beispielsweise seien 100 Festkommazahlen $a_1, a_2, \ldots, a_{100}$ im Wortformat gespeichert. Diese Zahlen sollen derart verarbeitet werden, daß jeweils ein anderer Wert a_i ($1 \leqq i \leqq 100$) adressiert wird. Für diese Methode der Adressierung kann bei den Befehlen im RX-Format ein *allgemeines Register als Indexregister* verwendet werden. Mit Hilfe des im Indexregister gespeicherten Wertes wird im Beispiel jeweils ein anderes a_i adressiert.
Zum besseren Verständnis soll der Zusammenhang zwischen Basisregister und Indexregister an einem Beispiel dargestellt werden.



Auf die Programmierung derartiger Aufgabenstellungen wird im Abschnitt 9.2. eingegangen.

3. Besonderheiten der Programmablaufplanung

Die Symbole für das Aufstellen von Programmablaufplänen (PAP) sind im DDR-Standard „Datenfluß- und Programmablaufpläne“ (TGL 22 451) festgelegt. Beim Einsatz des Betriebssystems DOS/ES kommen spezifische Verarbeitungsweisen zur Wirkung, die im maschinenorientierten Programmablaufplan berücksichtigt werden sollten.

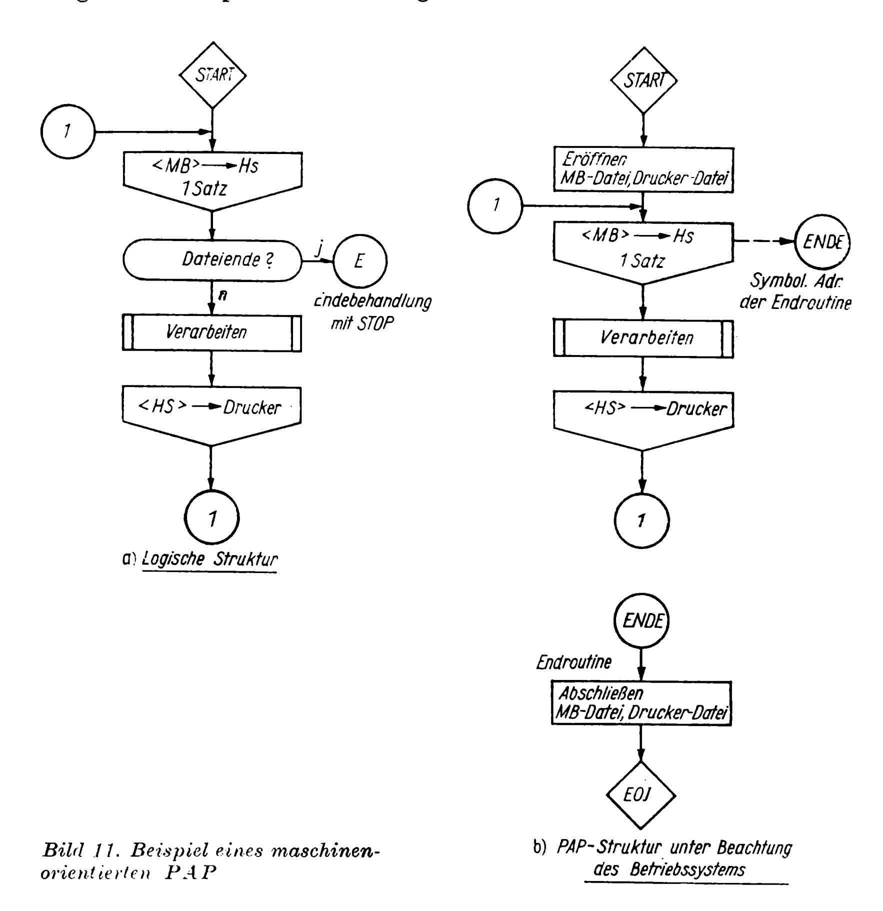


Bild 11. Beispiel eines maschinenorientierten PAP

So verzweigt beispielsweise eine Eingaberoutine (GET-Makro) des Betriebssystems *automatisch* zu einer Dateienderoutine, wenn bei einer Eingabe das Eingabeende erkannt wird. Diese automatische Verzweigung kann im PAP durch eine gestrichelte Linie dargestellt werden.

Die bisher bekannte Verfahrensweise, daß nach einer Eingabe von Daten abgefragt wird, ob das Dateiende vorliegt, *entfällt*, wenn mit den Standardroutinen des Betriebssystems gearbeitet wird (Normalfall).

Eine weitere Eigenheit ist das Eröffnen (OPEN) bzw. Schließen (CLOSE) von Dateien. Im PAP sollte dies mit dargestellt werden. Die in TGL 22 451 angegebene Zeichenfolge STOPP für das Ende der Programmabarbeitung kann durch die Zeichenfolge EOJ (end of job), die gleichzeitig den Operationskode den entsprechenden Makros darstellt, ersetzt werden. Die Zeichenfolge STOPP ist deshalb ungünstig, weil die EDVA durch die Eigenart der Steuerung bei Programmende nicht in den Stop-Zustand übergeht, sondern sofort die Abarbeitung eines nächsten Programms beginnt (Stapelverarbeitung).

Beispiel

Es ist ein maschinenorientierter PAP aufzustellen. Eine Magnetbanddatei soll satzweise eingelesen werden. Die eingelesenen Sätze sollen verarbeitet (arithmetische Operationen) und ausgedruckt werden (Bild 11). Der Zyklus Eingabe—Verarbeitung—Ausgabe wird so lange durchlaufen, bis die Eingaberoutine des Betriebssystems das Dateiende feststellt. Es erfolgt ein automatisches Verzweigen zur Endroutine (Bild 11b).

4. Assembler

4.1. Übersicht

Im Rahmen des Betriebssystems DOS/ES können Programme in verschiedenen Programmiersprachen z. B. Assemblersprache, RPG, PL/1 u. a. geschrieben werden. Diese als *Ursprungsmoduln* bezeichneten Programme werden vom entsprechenden Sprachübersetzer in einen *Objektmodul* übersetzt. Während die Übersetzung eines in RPG oder PL/1 geschriebenen Programms ein sog. *Compiler* übernimmt, werden Assemblerprogramme vom Sprachübersetzer für die Assemblersprache, kurz *Assembler*, übersetzt. Somit wäre prinzipiell zwischen der Assemblersprache und dem dazu notwendigen Übersetzungsprogramm, dem Assembler, zu unterscheiden.

4.1.1. *Assemblersprache*

Die Assemblersprache ist eine symbolische maschinenorientierte Programmiersprache, die sich den Eigenheiten der EDVA am besten anpaßt. In der Assemblersprache sind die Elemente eines symbolischen Programms sowie die Regeln und Vorschriften für dessen Aufbau festgelegt. So z. B. werden die Befehle symbolisch kodiert; eine Verwendung von symbolischen Adressen ist gegeben.

Die Assemblersprache wird in

Basis[assembler]sprache und

Makro[assembler]sprache

untergliedert. Die Makrosprache ist eine Erweiterung der Basissprache und setzt diese voraus. Im Rahmen des vorliegenden Bandes soll nur auf die Basissprache eingegangen werden.

4.1.2. *Assembler*

Die Aufgabe des Assemblers besteht in der Umwandlung des Ursprungsmoduls in einen Objektmodul. Dabei kann der Ursprungsmodul über die Datenträger Lochkarte, Magnetband oder Magnetplatte eingelesen werden. Das Ergebnis des Assemblerlaufs stellt den Objektmodul dar, der wahlweise auf Lochkarte, Magnetband oder Magnetplatte ausgegeben werden kann. Bei der Übersetzung wird ein Assemblerprotokoll hergestellt. In diesem sind die symbolischen Programmelemente (Befehle, symbolische Adressen u. a.) ihren internen (übersetzten) Äquivalenten gegenübergestellt. Während des Assemblerlaufs wird eine Prüfung des Ursprungsmoduls auf syntaktische Fehler durchgeführt. Diese Fehler werden im Assemblerprotokoll mit aufgelistet.

4.1.3. *Entwicklungsstufen eines Programms*

Das Ergebnis des Assemblerlaufs, der Objektmodul, kann noch nicht abgearbeitet werden. Ein weiterer Arbeitsschritt, das Verbinden des Objektmoduls durch den Programmverbinder zum Lademodul (Phase),

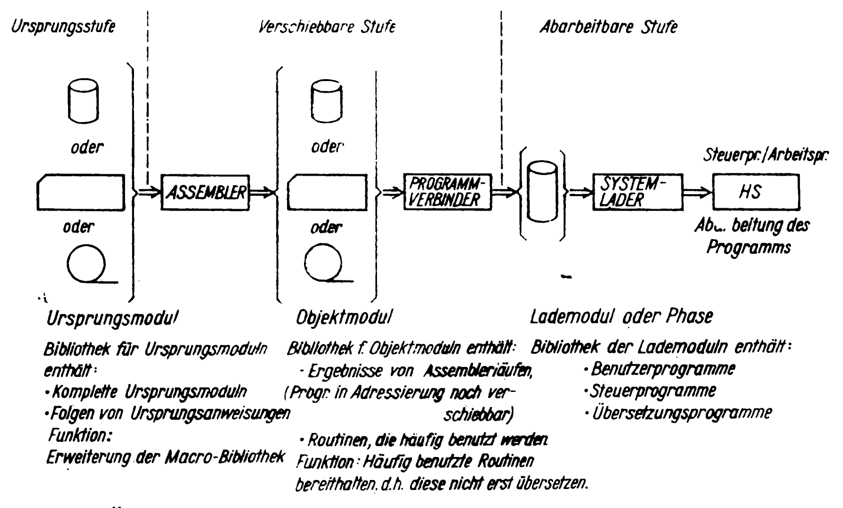


Bild 12. *Übersicht über die Entwicklungsstufen eines Programms [4]*

ist notwendig. Im Bild 12 sind die Etappen vom Ursprungsmodul bis zum abarbeitbaren Programm, dem Lademodul, zusammengefaßt. Die verwendeten Begriffe sollen kurz erläutert werden:

Ursprungsmodul. Die Gesamtheit der symbolischen Anweisungen, die dem Assembler in einem Übersetzungslauf eingabeseitig übermittelt werden, bilden den Ursprungsmodul.

Objektmodul. Der Objektmodul ist das Ergebnis des Assemblerlaufs. Das Programm ist noch nicht abarbeitbar. Durch den Programmverbinder wird der Lademodul (die Phase) hergestellt.

Lademodul (Phase). Die Phase ist das Ergebnis des Programmverbinderlaufs. Sie wird in den Hauptspeicher geladen und abgearbeitet.

4.2. Formaler Aufbau der symbolischen Anweisungen

4.2.1. *Programmformular*

Das Programmformular (Bild 13) dient zum Schreiben des symbolischen Programms. Die Befehls- bzw. Anweisungsfolge wird in das Programmformular eingetragen. Dabei wird eine Anweisung bzw. ein Befehl in eine Formularzeile eingetragen. Falls eine Formularzeile für die Eintragungen nicht ausreicht, ist eine Fortsetzungszeile (ab Spalte 16)

Programm		Lochhinweise	Blatt von
		Zeichen	Programmformular Assembler
Programmierer	Datum	Lochung	

Name 1 8	Operation 10 14	Operand und Bemerkungen 25 30 35 40 45 50 55 60 65 71	Problem Folge 73 75 80

Bild 13. Programmformular

gestattet. Dabei muß in der Erstzeile (in Spalte 72) ein vom Leerzeichen verschiedenes Zeichen (z. B. $\ast$) eingetragen werden, welches angibt, daß diese Anweisung in der folgenden Zeile fortgesetzt wird. Die symbolischen Anweisungen können aus verschiedenen Feldeintragungen wie Namensfeld, Operationsfeld, Operandenfeld und Bemerkungsfeld bestehen. Die einzelnen Felder sollen kurz beschrieben werden:

Namensfeld. Im Namensfeld kann ein Symbol eingetragen werden. Dieses stellt die symbolische Adresse der zugehörigen Anweisung oder des zugehörigen Befehls dar.

Operationsfeld. Im Operationsfeld wird der Operationskode der symbolischen Anweisung eingetragen.

Operandenfeld. Im Operandenfeld werden Hauptspeicheradressen, Registernummern usw. eingetragen. Treten in einer symbolischen Anweisung mehrere Operanden auf, sind diese durch Komma zu trennen.

Bemerkungsfeld. Im Bemerkungsfeld kann eine symbolische Anweisung kommentiert werden. Alle die Zeichen, die dem Operandenfeld durch mindestens eine Leerspalte getrennt folgen, interpretiert der Assembler als Bemerkung. Bemerkungen werden im Übersetzungsprotokoll mit aufgelistet.

Im allgemeinen wird bei der Programmierung diese starre Einteilung verwendet. Sie erhöht die Übersicht. Es ist aber auch möglich, symbolische Anweisungen im *freien Format* (ohne Feldeinteilung) zu schreiben. Generell gilt, daß die einzelnen Felder durch mindestens ein Leerzeichen getrennt werden und die Reihenfolge Name, Operation, Operand, Bemerkung einzuhalten ist. Wenn die symbolischen Anweisungen im starren Format gemäß Programmformular geschrieben werden, haben die Eintragungen in die Felder linksbündig zu erfolgen.

Bemerkungen: Jede Zeile des Programmformulars ist in eine Lochkarte abzulochen. Dabei ist die Einteilung entsprechend dem Programmformular beizubehalten. Das Problemfolgefeld (Spalten 73 bis 80) kann in den Spalten 73 bis 76 mit der Problemkennzeichnung und in den Spalten 77 bis 80 mit einer Folgenumerierung belegt werden. Diese Eintragungen werden mit abgelocht und können zur Prüfung auf aufsteigende Folge der Lochkarten bei der Eingabe des Ursprungsmoduls dienen. Die Gesamtheit der in Lochkarten abgelochten symbolischen Anweisungen bzw. Befehle bildet einen Eingabedatenbestand (Eingabedatei) für einen Assemblerlauf. Die Arbeit des Assemblers besteht nun darin, jeder symbolischen Anweisung ein ihr entsprechendes internes Äquivalent zuzuordnen. Dabei wird der Eingabedatenbestand sequentiell, also Anweisung für Anweisung, übersetzt.

4.2.2. *Elemente der Basissprache*

Im folgenden soll zwischen einem *einfachen Ausdruck (Term)* und einem *zusammengesetzten Ausdruck* unterschieden werden. Zusammengesetzte Ausdrücke entstehen durch Kombination von Termen mit den Operatoren +, −, * und /. Dabei ist unter anderem zu beachten, daß ein derartiger Ausdruck nicht mit einem Operator beginnen darf. Im folgenden sollen die Terme näher betrachtet werden. Jeder Operand eines Befehls oder einer Assembleranweisung wird durch einen Ausdruck gebildet, dem bei der Übersetzung ein Wert zugeordnet wird. In der Assemblersprache werden fünf Arten von Termen unterschieden:

Direktwert
Symbol
Bezugnahme auf den Speicherplatzzähler (Sternadresse)
Literal
Bezugnahme auf das Längenattribut (Längenanzeiger).

4.2.2.1. *Direktwerte*

Direktwerte stellen einen absoluten Wert dar. Bei der Übersetzung wird ihnen kein neuer Wert zugewiesen. Sie werden verwendet, um Registernummern, absolute Adressen, Verschiebungen u. a. anzugeben. Es können vier Arten von Direktwerten verwendet werden:

dezimaler Direktwert
hexadezimaler Direktwert
binärer Direktwert
Zeichendirektwert.

Ein *dezimaler Direktwert* wird als vorzeichenlose Dezimalzahl angegeben. *Hexadezimale Direktwerte* werden durch ein vorangestelltes X gekennzeichnet. Die folgenden Hexadezimalziffern sind in Apostrophe einzuschließen. *Binäre Direktwerte* werden durch ein vorangestelltes B gekennzeichnet. Die folgenden Binärziffern (0 oder 1) sind in Apostrophe einzuschließen. *Zeichendirektwerte* kennzeichnet man durch ein vorangestelltes C. Die folgenden Zeichen sind durch Apostrophe einzuschließen.

Beispiel

```
1. Dezimaler Direktwert
         MVC   FELD(8),ERG
    oder MVC   2Ø(8,3),6Ø(3)

2. Hexadezimaler Direktwert
         TM    BER,X'8Ø'

3. Binärer Direktwert
         TM    BER,B'1ØØØØØØØ'

4. Zeichendirektwert
         CLI   ADR,C'A'
```

Die eigentlichen Befehlswirkungen werden bei der Behandlung der Befehle erläutert. Hier soll nur auf die Operanden im Zusammenhang mit Direktwerten eingegangen werden. Im ersten Beispiel stellt FELD(8) den ersten Operanden dar, wobei die Operandenlänge 8 Bytes beträgt. In der folgenden Zeile wird Operand 1 durch 20(8,3) und Operand 2 durch 60(3) dargestellt. Alle Angaben stellen Direktwerte dar. Im zweiten, dritten und vierten Beispiel sind Befehle im SI-Format (Abschn. 2.6.) dargestellt. Die angegebenen Direktwerte sind Bestandteil der Befehle.

4.2.2.2. *Symbole*

Symbole werden dazu verwendet, um Speicherplätze des HS ohne Kenntnis deren physischer Lage (echte, numerische Adresse) zu adressieren (symbolische Adressierung). Es ist Aufgabe des Assemblers, die entsprechenden Voraussetzungen für die Zuordnung der symbolischen zu den physischen HS-Adressen zu schaffen. Dazu ordnet der Assembler unter anderem während der Übersetzung des Ursprungsmoduls jedem auftretenden Symbol ein Längen- und ein Adreßattribut zu.
Das *Längenattribut* ergibt sich aus der Länge des Feldes, das durch die vom Symbol benannte Anweisung im Objektmodul belegt wird. Beispielsweise benennt ein Symbol KONST eine Anweisung, die nach der Übersetzung, also im Objektmodul, eine Länge von 8 Bytes belegt. Das entsprechende Längenattribut wäre hierbei 8.
Das *Adreßattribut* entspricht dem aktuellen Speicherplatzzählerstand. Unter *Speicherplatzzähler* soll ein vom Assembler geführter Zähler verstanden werden. Dieser Zähler wird vom Assembler jeweils um den Wert erhöht, dem die momentan zu übersetzende symbolische Anweisung in ihrer internen Form entspricht. Im allgemeinen beginnt die Speicherplatzzuordnung bei der Übersetzung des symbolischen Programms in den Objektmodul mit dem Zählerstand Null. Wird nun beispielsweise eine Anweisung übersetzt, die intern eine Länge von 6 Bytes benötigt, erhöht der Assembler den Speicherplatzzähler um 6 usw.
Bei der Verwendung von Symbolen ist folgendes zu beachten:

Ein Symbol kann ein bis acht Zeichen umfassen.
Zulässige Zeichen sind die Buchstaben A bis Z, die Ziffern 0 bis 9 und die drei Sonderzeichen @, #, $.
Das erste Zeichen eines Symbols muß ein Buchstabe sein.
Leerzeichen innerhalb des Symbols sind unzulässig.

Beispiel

Name	Operation	Operand
Gültige Symbole:		
ZYKLUS1		
ANFANG		
ADR11		
KE#TTE		
Ungültige Symbole		
ADR 1		Leerzeichen im Symbol
ANF.-ADR		Unzulässige Sonderzeichen

Übung:

Ü. 4.1. Welche Symbole werden vom Assembler als fehlerhaft erkannt?

a) ZYKLUS

b) BEZ1.1

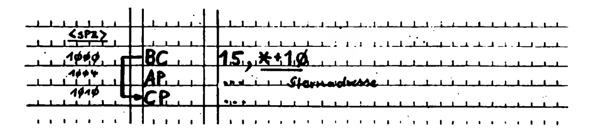
c) ANF

d) TEILE.-NR.

4.2.2.3. *Bezugnahme auf den Speicherplatzzähler (Sternadresse)*

Mit Hilfe des *Speicherplatzzählers* werden Befehlen und Datendefinitionen lückenlos HS-Plätze zugeordnet. Beim Programmieren kann man sich durch Angabe eines Sternes (∗) im Operandenfeld auf den jeweils aktuellen Speicherplatzzählerstand beziehen.

Beispiel



4.2.2.4. *Literale*

Literale sind Konstanten, die durch ihre Eintragung im Feld „Operand" eines symbolischen Befehls definiert werden. Gekennzeichnet werden Literale durch ein Gleichheitszeichen, dem die Datendefinition (s. Konstantendefinitionen, Abschn. 4.3.4.) folgt. Bei der Verwendung von Literalen ist folgendes zu beachten:

Literale dürfen nur in symbolischen Befehlen auftreten, die sich auf HS-Plätze beziehen (nicht im RR-Format und SI-Format).

Im Befehl ist nur ein Literal gestattet.

Ein Literal darf nicht mit anderen Termen kombiniert werden, d. h., das Literal stellt allein einen Operanden dar.

Beispiel

```
                      Direktwert (dezimal)
         MVC   ADR(3),=C'ABC'
                      Literal
   oder  LM    3,5,=A(FELD,5,FELD+99*5)
                      Adreßliteral
```

Im ersten Fall wird die Zeichenfolge ABC als Literal angesehen. Diese Zeichen werden im HS gespeichert und für das Programm bereitgestellt. Der zweite Fall zeigt die Verwendung eines Adreßliterals. Da Literale grundsätzlich (bis auf das einleitende Gleichheitszeichen) das Format der entsprechenden Konstanten aufweisen, wird auf diese verwiesen (Abschn. 4.3.4.).

4.2.2.5. *Bezugnahme auf das Längenattribut (Längenanzeiger)*

Längenanzeiger beziehen sich auf das Längenattribut eines Symbols. Sie sind gekennzeichnet durch L' vor dem Symbol. Eine Anwendung ist aus Abschn. 5.1. zu entnehmen.

4.3. Assembleranweisungen

Unter Assembleranweisungen sollen alle Anweisungen an den Assembler verstanden werden. Im Unterschied zu den Befehlen sind die Mehrzahl der Assembleranweisungen nur während der Übersetzung des Ursprungsmoduls wirksam. Konstanten- und Speicherplatzdefinitionen liefern echte Beiträge beim Aufbau des Objektmoduls. Assembleranweisungen können wie folgt gruppiert werden:

Anweisungen zur Programmverbindung und -teilung
(START, CSECT, DSECT, ENTRY, EXTRN, COM)

Anweisungen zur Steuerung des Assemblerlaufs
(ICTL, ISEQ, PUNCH, REPRO, LTORG, CNOP, COPY, ORG, END)

Anweisungen für die Protokollierung
(TITLE, EJECT, SPACE, PRINT)

Basisregisteranweisungen
(USING, DROP)

Definitionsanweisungen
(EQU, DS, DC, CCW).

Im Rahmen des vorliegenden Bandes kann nur auf die wichtigsten Anweisungen eingegangen werden. Eine vollständige Beschreibung ist in [5] zu finden. Bei der weiteren Darstellung werden für wahlfrei verwendbare Ausdrücke eckige Klammern [...] verwendet. Damit soll ausgesagt werden, daß der Ausdruck verwendet oder auch weggelassen werden kann.

4.3.1. *Anweisungen START, END*

START-Anweisung

```
[symbol]    START   [direktwert]
     z.B.
            START
```

Wirkung der Anweisung. START benennt, sofern ein Symbol angegeben, den ersten (oder einzigen) Programmteil eines symbolischen Programms. Der Speicherplatzzähler wird auf den angegebenen Wert eingestellt. In der Programmierpraxis wird dieser Wert meist weggelassen. In diesem Fall beginnt der Speicherplatzzähler beim Stand Null. Die START-Anweisung ist die erste Anweisung des symbolischen Programms.

END-Anweisung

```
            END     [verschiebbarer Ausdruck]
     z.B.
            START                 Anf. des symb. Programms
ANFANG      BALR
            :
            END     ANFANG        Ende des symb. Programms
```

Wirkung der Anweisung. END zeigt dem Assembler das Ende des Ursprungsmoduls an. Sie ist die letzte Anweisung des symbolischen Programms. Der Operand bezeichnet den Befehl, der nach dem Laden der Phase als erster ausgeführt werden soll.

4.3.2. *Bereichsdefinitionen*

Durch Bereichsdefinitionen werden Hauptspeicherbereiche für das Programm bereitgestellt. Diese Bereiche werden durch Definitionen nicht gelöscht.

Format der Anweisung

```
[symbol]    DS      vtLn
```

Die Operandenangaben haben folgende Bedeutung:
Vielfachschlüssel v gibt die Anzahl der zu reservierenden Felder an. Wenn v nicht angegeben ist, wird v = 1 angenommen.
Typenschlüssel t gibt die Art des zu definierenden Feldes an. Für t können folgende Zeichen verwendet werden:

t =	Bedeutung
C	Zeichen (Bytes)
H	Halbwort
F	Wort
D	Doppelwort

Längenschlüssel Ln gibt die Länge des zu reservierenden Feldes an.

Wirkung der Anweisung

Durch die DS-Anweisung wird ein HS-Bereich reserviert. Diesem Bereich wird das im Namensfeld verwendete Symbol zugeordnet.

Beispiel

1. Für eine Lochkarteneingabe soll ein Bereich von 80 Bytes mit dem Namen LKEIN definiert werden.

```
LKEIN  DS  CL8Ø
```

Dem Bereich LKEIN wird vom Assembler das Längenattribut **80** zugeordnet. Der Vielfachschlüssel v wird mit 1 angenommen. Somit wird im Programm unter der symbolischen Adresse LKEIN ein Speicherbereich von 1 * 80 Bytes frei gehalten.

2. Oft ist es sinnvoll, sog. „Unterfelder" zu verwenden. Eine Lochkarte habe folgenden Aufbau:

Spalten 1 bis 6	Kundennummer
Spalten 7 bis 30	Adresse des Kunden
Spalten 31 bis 37	Bestellmenge.

(Die weiteren Angaben sollen nicht interessieren.)

Die Namen der Unterfelder sollen wie folgt gewählt werden:

EINB	Name des LK-Eingabebereichs
KNR	Feld für Kundennummer
ADR	Feld für Adresse
MENGE	Feld für Bestellmenge.

Die Struktur des Eingabebereichs kann wie folgt realisiert werden:

```
EINB   DS  ØCL37
KNR    DS  CL6
ADR    DS  CL24
MENGE  DS  CL7
```

Unter der symbolischen Adresse EINB wird kein Speicherplatz reserviert (v = 0, also 0 * 37 Bytes). Dem Symbol EINB wird aber das unter Ln angegebene Längenattribut 37 zugeordnet. Die eigentliche Bereichsdefinition erfolgt durch die Unterfelder.

Symbolische Adresse	Längenattribut	Rerservierter Bereich (Bytes)
EINB	37	0
KNR	6	6
ADR	24	24
MENGE	7	7

Zu beachten ist, daß die Summe der Bereichslängen der Unterfelder gleich der Länge des übergeordneten Feldes (EINB) sein muß.

3. Im Abschn. 2.3. wurde auf die Verwendung integraler Grenzen hingewiesen. Es soll ein Speicherbereich für 100 Dualzahlen im Wortformat unter der symbolischen Adresse BER definiert werden.

```
BER      DS    100F
```

Wirkung der Anweisung

Es wird unter der symbolischen Adresse BER ein Bereich von 100 ∗ 4 Bytes (Wortformat) frei gehalten.
Der Speicherplatzzähler wird von der Bereichszuordnung auf eine durch 4 teilbare Adresse (integrale Grenze) gestellt.
Damit ist garantiert, daß die Adresse BER den Forderungen der Binärarithmetik (Abschn. 8.2.) genügt.
Der symbolischen Adresse BER wird das Längenattribut 4 zugeordnet.

4. Es soll unter der symbolischen Adresse KONVERT ein 8-Bytes-Feld definiert werden. Dieses Feld soll an einer durch 8 teilbaren Adresse beginnen.

```
KONVERT  DS    D
```

Wirkung der Anweisung

KONVERT beginnt an einer durch 8 teilbaren Adresse.
Es wird ein Feld in der Länge von 1 ∗ 8 Bytes (Doppelwort) reserviert.
Das Längenattribut von KONVERT beträgt 8 Bytes.

Es kann folgendes verallgemeinert werden:

1. Das einem Symbol zugeordnete Längenattribut ergibt sich entweder aus der unter Ln angegebenen Feldlänge oder aus dem Typ des Feldes. Werden Längenschlüssel Ln und Typenschlüssel t in einer Anweisung verwendet, hat die unter Ln getroffene Angabe höhere Priorität.
2. Bei Angabe eines Längenschlüssels Ln wird der Speicherplatzzähler nicht auf integrale Grenzen ausgerichtet.
3. Die Verwendung des Vielfachschlüssels v hat folgende Wirkung:

v = 0	Kein Bereich frei gehalten
v = n	Bereich wird n-mal frei gehalten
v nicht belegt	Assembler nimmt v = 1 an.

Übung

Ü. 4.2. Es ist ein Bereich von 50 Bytes zu definieren. Der Name des Bereichs sei EIN. Folgende Unterfelder sollen vereinbart werden:

TEILENR	7
BEZEICH	39
MENGE	4

Ü. 4.3. Unter der symbolischen Adresse ZWSP soll ein 8-Bytes-Feld definiert werden (Doppelwortgrenze).

4.3.3. *Anweisung USING*

Im Abschn. 2.7. wurden die Begriffe „Basisregister" und „Verschiebung" eingeführt und erläutert. Zum Zeitpunkt des Assemblerlaufs werden die symbolischen Adressen in eine Basisregisternummer und eine Verschiebung zerlegt. Dabei muß dem Assembler bekannt sein, welche(s) Register als Basisregister verwendet werden soll(en) und welcher Wert als Basisadresse für die Errechnung der relativen Adressen (Verschiebungen) zu benutzen ist. Diese Mitteilungen an den Assembler übernimmt die Anweisung USING („Zuweisen Basisregister").

```
        USING  a,r1,...,r16
z.B.    USING  *,5
```

Die Operandenangaben haben folgende Bedeutung:

a	absoluter oder verschiebbarer Adreßausdruck, wird vom Assembler als Basisadresse verwendet
$r_1, \ldots, r_{16}$	absolute Ausdrücke (0 bis 15).

Wirkung der Anweisung

1. Der Ausdruck a gibt den Wert an, der als Basisadresse zur Errechnung der relativen Adressen (Verschiebungen zu a) zu verwenden ist (z. B. $*$, d. h. aktueller Speicherplatzzählerstand).

2. Der zweite Operand bezeichnet das (die) allgemeine(n) Register, das (die) als Basisregister zu verwenden ist (sind).

Bemerkung. Unter Berücksichtigung der Routinen des Betriebssystems DOS/ES sollten die allgemeinen Register 3 bis 12 als Basisregister verwendet werden.

3. Bei Angabe mehrerer Register wird dem Assembler mitgeteilt, daß das Register r_1 die Basisadresse $a + 0 * 4096$ und die Register $r_2, r_3 \ldots$ die Basisadressen $a + 1 * 4096, \ldots$ enthalten sollen.

Bemerkung. Das angegebene Basisregister wird durch USING nicht geladen. Das Laden des Basisregisters erfolgt durch den Befehl BALR (Abschn. 7.3.3.) zum Zeitpunkt der Programmabarbeitung.

Die Anweisung

```
        USING  *,5
```

erklärt das allgemeine Register 5 als Basisregister. Damit hat dieses Register eine besondere Funktion erhalten und kann i. allg. nicht mehr für andere Zwecke im Programm verwendet werden. Als Basisadresse wird der aktuelle Speicherplatzzählerstand verwendet. Der Assembler

bestimmt aus der Differenz von effektiver Adresse des betrachteten Symbols und spezifizierter Basisadresse die entsprechende Verschiebung.

Beispiel 2

<Speicherplatz-zähler>			
ØØØ		START	
:	ANF	BALR	5,Ø
:		USING	*,5
		:	
		MVC	AKNR,EKNR
		:	
32Ø	EKNR	DS	CL4
324	AKNR	DS	CL4
		:	
		END	ANF

Der Befehl BALR belegt im Objektmodul 2 Bytes. Somit ergibt sich $SPZ_{aktuell}$ für USING: 002.
Die Operandenadressen im MVC-Befehl sind intern, also im Objektmodul, wie folgt belegt:

Op'K.	L0	B1	D1	B2	D2
		5	322	5	318

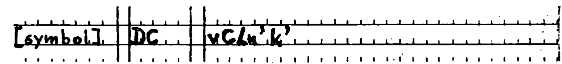
B1, D1: Zerlegung des Operanden AKNR
B2, D2: Zerlegung des Operanden EKNR

4.3.4. *Konstantendefinitionen*

Durch Konstantendefinitionen werden konstante Daten in das Programm eingeführt. Der Operationskode lautet DC. Im Namensfeld kann ein Symbol eingetragen werden. Dieses Symbol wird zur Adressierung der Konstanten verwendet.
Es sind folgende Arten von Konstanten zu unterscheiden:
Fest- und Gleitkommakonstanten
dezimale, binäre und hexadezimale Konstanten
Zeichen- und Adreßkonstanten.

4.3.4.1. *Zeichenkonstante*

Format



Operanden

v	Vielfachschlüssel	C	Typenschlüssel (Zeichenkonstante)
Ln	Längenschlüssel	k	Konstante (in Apostrophe eingeschlossen)

Wirkung der Anweisung. Die angegebene Konstante k wird für das Programm bereitgestellt. Dabei wird der Konstanten eine HS-Adresse zugeordnet. Zur Adressierung dieser Konstanten wird das im Namensfeld eingetragene Symbol (symbolische Adresse) verwendet.

Bemerkung. Es sind alle Zeichen des Arbeitskodes (Abschn. 1.1.1.) zugelassen. Treten innerhalb der Zeichenkonstanten die Zeichen ' (Apostroph) und & (Ampersand) auf, sind diese jeweils zweimal zu schreiben, wenn sie in der internen Form einmal erscheinen sollen. Jedes Zeichen der angegebenen Konstante belegt ein Byte im HS. Unter dem angegebenen Symbol wird die Konstante linksbündig gespeichert. Stimmen implizite Länge (entspricht der internen Zeichenzahl der Konstante) und explizite Länge (Angabe durch Ln) nicht überein, werden die rechten Zeichen der Konstanten weggelassen (wenn Konstante zu lang) bzw. mit Leerzeichen (X'40') aufgefüllt (wenn Konstante zu kurz). Die Konstante bleibt linksbündig erhalten. Maximale Konstantenlänge (intern): 256 Bytes.

Beispiel

KONST1	DC	C'ROBOTRON 21'
KONST2	DC	2C'ROBOTRON'
KONST3	DC	2CL3'ROBOTRON'
KONST4	DC	CL10'ROBOTRON'

Ergibt im HS:

KONST1 → ROBOTRONb̸21

KONST2 → ROBOTRONROBOTRON

KONST3 → ROBROB

KONST4 → ROBOTRONb̸b̸

Anmerk.: b̸ ≙ X'40' (Leerzeichen)

4.3.4.2. *Binärkonstante*

Format

[symbol]	DC	vBLn'k'

Operanden. Vgl. Zeichenkonstante, aber B Typenschlüssel (Binärkonstante).

Wirkung der Anweisung. Vgl. Zeichenkonstante.

Bemerkung. Durch die Binärkonstante ist es möglich, konstante Daten bitweise zu definieren. Als Zeichenvorrat innerhalb der Konstanten darf hier 0 und 1 auftreten. Je acht Binärziffern werden in einem Byte intern dargestellt. Ist die Anzahl der angegebenen Binärziffern nicht durch 8 teilbar, fügt der Assembler links soviel binäre Nullen an, bis eine durch 8 teilbare Anzahl von Binärziffern erreicht ist. Ein durch Ln begründetes Auffüllen bzw. Weglassen von Ziffern der angegebenen Konstante erfolgt links. Die Konstante bleibt rechtsbündig erhalten.

Maximale Konstantenlänge (intern): 256 Bytes.

Beispiel

Name	Operation	Operanden
K1	DC	B'11Ø'
K2	DC	BL1'111Ø11Ø111'

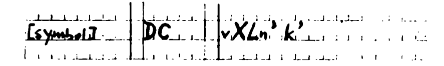
Ergibt im HS:

K1 → ØØØØØ11Ø

K2 → 1Ø11Ø111

4.3.4.3. *Hexadezimalkonstante*

Format



Operanden. Vgl. Zeichenkonstante, aber X Typenschlüssel (Hexadezimalkonstante).

Wirkung der Anweisung. Vgl. Zeichenkonstante.

Bemerkung. Der Zeichenvorrat der Konstante ergibt sich aus den Hexadezimalziffern 0 bis 9 und A bis F. Jede Hexadezimalziffer wird in ihr Binäräquivalent umgewandelt und so in einem Halbbyte gespeichert. Bei einer ungeraden Anzahl von Hexadezimalziffern wird das linke Halbbyte mit X'0' aufgefüllt. Ein durch Ln begründetes Auffüllen bzw. Weglassen von Ziffern der angegebenen Konstante erfolgt links. Die Konstante bleibt rechtsbündig erhalten.

Maximale Konstantenlänge (intern): 256 Bytes.

Beispiel

Name	Operation	Operanden
K3Ø	DC	X'ØA'
K3	DC	X'31C'

Ergibt im HS:

K3Ø → ØØØØ1Ø1Ø

ØØØØ ØØ11 ØØØ1 11ØØ (↑K3, ↑K3+1)

4.3.4.4. *Dezimalkonstanten*

Dezimalkonstanten sind in gepackter Form (Typenschlüssel P) und ungepackter Form (Typenschlüssel Z) möglich. Es können mehrere Konstanten durch Komma getrennt in einer Definitionsanweisung angegeben werden. Die Konstante kann vorzeichenbehaftet sein. Bei fehlendem Vorzeichen nimmt der Assembler Plus an. Die Speicherungsform von gepackten bzw. ungepackten Daten wurde im Abschn. 2.2. erläutert. Ein in der Konstante angegebener Dezimalpunkt wird vom Assembler ignoriert.

Format

```
Gepackte Konstante:
[symbol]      DC     v,PLn'k1,k2,...,kn'
Ungepackte (gezonte) Konstante:
[symbol]      DC     v,ZLn'k1,k2,...,kn'
```

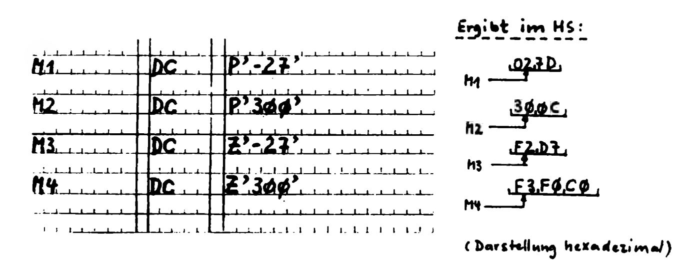
Operanden

v, Ln	vgl. Zeichenkonstante
P, Z	Typenschlüssel (P gepackte Konstante, Z ungepackte Konstante)
$k_1, \ldots, k_n$	mehrere Konstanten k (durch Komma getrennt)

Wirkung der Anweisung. Die angegebene(n) Konstante(n) wird (werden) für das Programm bereitgestellt. Dabei erhält jede Konstante ihr entsprechendes Vorzeichen.

Bemerkung. Die Konstante wird rechtsbündig in ihr zugeordnetes Feld eingetragen. Stimmen implizite und explizite Länge nicht überein, werden links Stellen weggelassen, oder es wird beim Typ P mit hexadezimalen Nullen und beim Typ Z mit X'FO' aufgefüllt.
Maximale Konstantenlänge (intern) für beide Typen: 16 Bytes.

Beispiel



4.3.4.5. *Festkommakonstanten*

Festkommakonstanten treten im Halbwortformat (Typenschlüssel H) und im Wortformat (Typenschlüssel F) auf. Es können in einer Definitionsanweisung mehrere Konstanten (durch Komma zu trennen) angegeben werden. Die Konstante kann vorzeichenbehaftet sein. Bei fehlendem Vorzeichen nimmt der Assembler Plus an. Die Konstante(n) wird (werden) als Dezimalzahl(en) in der Definitionsanweisung angegeben. Der Assembler wandelt diese in ihr Binäräquivalent um und stellt diese Binärzahl(en) entsprechend dem Typenschlüssel im Halbwort- oder Wortformat (Abschn. 2.3.) bereit.

Format

```
Konstante im Halbwortformat
[symbol]   DC   vHLnSnEn'k1,...,kn'
Konstante im Wortformat
[symbol]   DC   vFLnSnEn'k1,...,kn'
```

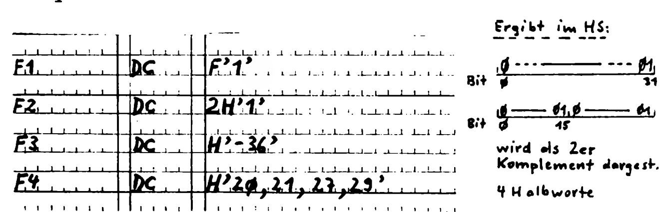
Operanden

v, Ln	vgl. Zeichenkonstante
H, F	Typenschlüssel (H Halbwortformat, F Wortformat)
$k_1, \ldots, k_n$	mehrere Konstanten k (durch Komma getrennt)
Sn	Maßstabschlüssel (berücksichtigt den gebrochenen Teil der angegebenen Konstanten)
En	Exponentenschlüssel

Wirkung der Anweisung. Die angegebene(n) Konstante(n) wird (werden) als Festkommazahl im Halbwort- oder Wortformat bereitgestellt.

Bemerkung. Ist kein Längenschlüssel angegeben, erfolgt ein Ausrichten der Speicheradressen auf integrale Grenzen (Abschn. 2.3.). Die Konstante wird rechtsbündig in ihr zugeordnetes Feld eingetragen. Bei Differenz zwischen expliziter und impliziter Länge wird links aufgefüllt bzw. weggelassen (Achtung: Vorzeichen!).

Beispiel



Übung

Ü. 4.4. Folgende Konstanten sind zu definieren:

a) Zeichenkonstanten
   UNGUELTIGER WERT
   1. 9. 1972

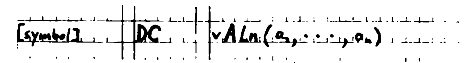
b) Dezimalkonstanten
   gepackt —70
   ungepackt 60

c) Festkommakonstanten
   Wort 37
   Halbwort —1

4.3.4.6. *Adreßkonstanten*

Adreßkonstanten treten im Typ A, Y und S auf. Dabei handelt es sich um HS-Adressen, die in Form von Konstanten definiert werden. Verwendet werden Adreßkonstanten beim Laden von Registern (Abschn. 9.3.1.) und bei Bezugnahmen zwischen Programmabschnitten unterteilter Programme. Die Adreßkonstante vom A-Typ soll nachstehend erläutert werden. Angaben zum Y- und S-Typ sind aus [5] zu entnehmen.

Format



Operanden

v, Ln vgl. Zeichenkonstante

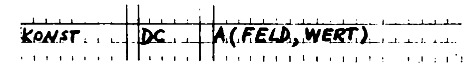
A Typenschlüssel (Adreßkonstante vom A-Typ)

$(a_1, a_2, \ldots, a_i, \ldots, a_{n-1}, a_n)$
mehrere Adreßkonstanten a_i (durch Komma getrennt)

Wirkung der Anweisung. Die in Klammern angegebenen Ausdrücke werden durch den Assembler als binäre Adreßwerte bereitgestellt.

Bemerkung. Die Adreßkonstante vom A-Typ belegt, wenn kein Längenschlüssel verwendet ist, 4 Bytes (Wortformat).

Beispiel



Unter der symbolischen Adresse KONST werden in je einem Wort die Adreßwerte für die symbolischen Adressen FELD und WERT binär gespeichert.

5. Transport- und Vergleichsoperationen

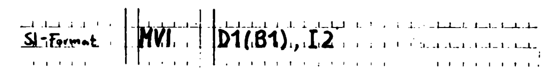
5.1.[1] Transportoperationen

In der Gruppe der Transportoperationen sollen Befehle dargestellt werden, die den

Transport eines Direktoperanden
Transport zwischen Hauptspeicherbereichen
Transport von Zonenbits bzw. numerischen Bits

realisieren.

Transport direktes Datenbyte



Befehlswirkung. Der im Befehl angegebene Direktoperand wird in das durch D1(B1) adressierte Feld übertragen.

Beispiel

Auf die Adresse ADR soll das Zeichen „A“ transportiert werden.

```
         MVI   ADR,C'A'
           :
ADR      DS    CL1
```

Bemerkung. HS-Plätze können auf diese Art mit definierten Werten belegt werden. Sie sind dann im Sinne eines Byteschalters zu verwenden (Abfragen mit CLI, Abschn. 5.2.).
Die folgenden Transportbefehle sind Befehle im logischen SS-Format. Die Verarbeitung erfolgt von links nach rechts. Dabei entscheidet die Länge des ersten Operanden (L0) über die Zahl der zu übertragenden Zeichen.
Die maximale Operandenlänge beträgt 256 Bytes.

Transport zwischen Hauptspeicherbereichen

```
SS-Form log.  MVC   D1(LØ,B1),D2(B2)
```

Befehlswirkung. Der Inhalt des durch D2(B2) adressierten Feldes wird in das durch D1(B1) adressierte Feld übertragen.

Beispiel

1. Der Inhalt des Feldes BER soll auf dem Feld ZWSP gespeichert werden.

```
         MVC   ZWSP,BER
           :
ZWSP     DS    CL4
BER      DS    CL4
```

2. Der Ausgabebereich ABER soll auf Leerzeichen (X'40') gelöscht werden.

```
         MVI   ABER,X'4Ø'                  Zyklisches
         MVC   ABER+1(79),ABER             Zeichensetzen
    oder
         MVI   ABER,C' '                   Arbeit mit
         MVC   ABER+1(L'ABER-1),ABER       Längenanzeiger
    oder
         MVC   ABER,=8ØC' '                Literal
           :
ABER     DS    CL8Ø
```

Bemerkung. Soll ein Feld auf binäre Nullen (X'00') gelöscht werden, kann der Befehl „Exklusives Oder“ (XC) (Abschn. 10.) angewandt werden.

Transport zwischen Hauptspeicherbereichen — Transport numerischer Bits

SS-Form. log. | MVN | D1(LØ,B1),D2(B2)

Befehlswirkung. Die numerischen Bits des durch D2(B2) adressierten Feldes werden in die Positionen der numerischen Bits des durch D1(B1) adressierten Feldes übertragen. Die Zonenbits beider Felder bleiben dabei unverändert.

Transport zwischen Hauptspeicherbereichen — Transport der Zonenbits

SS-Form. log. | MVZ | D1(LØ,B1),D2(B2)

Befehlswirkung. Die Zonenbits des durch D2(B2) adressierten Feldes werden in die Positionen der Zonenbits des durch D1(B1) adressierten Feldes übertragen. Die numerischen Bits beider Felder bleiben dabei unverändert.

5.2. Logische Vergleichsoperationen

In die Gruppe der hier zu besprechenden Vergleichsoperationen fällt der

logische Vergleich mit einem Direktoperanden und der

logische Vergleich zwischen Hauptspeicherinhalten.

Die sonstigen Vergleiche wie

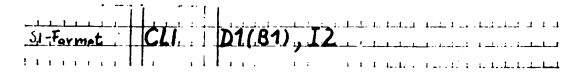
Vergleich von Dezimalzahlen und

Vergleich von Binärzahlen

werden in den Abschnitten 6.2.3. und 8.2.4. behandelt.
Prinzipiell interessiert bei den Vergleichsoperationen, ob der erste Operand größer, gleich oder kleiner als der zweite Operand ist. Diese Aussage, also das Vergleichsergebnis, bezieht sich auf das Primat des ersten Operanden. Das Vergleichsergebnis wird als Wirkung des Vergleichsbefehls im Programmstatuswort gespeichert. Beim Programmieren bezieht man sich auf diese gespeicherte Vergleichsaussage (Abschn. 7.).

Der logische Vergleich erfolgt bitweise von links nach rechts. Er endet bei der ersten Bitungleichheit.

Logischer Vergleich mit einem Direktoperanden



Befehlswirkung. Der im Befehl angegebene Direktoperand wird mit dem durch D1(B1) adressierten Feldinhalt verglichen.

Beispiel

Es soll abgefragt werden, ob auf der Adresse ADR das Zeichen „A“ gespeichert ist (Byteschalter).

```
         CLI   ADR,C'A'
         :     Auswerten des Vergleichs durch Sprungbefehl
ADR      DS    CL1
```

Logischer Vergleich zwischen Hauptspeicherinhalten

```
SS-Form.-log.  CLC   D1(LØ,B1),D2(B2)
```

Befehlswirkung. Der Inhalt des durch D1(B1) adressierten Feldes wird mit dem Inhalt des durch D2(B2) adressierten Feldes verglichen.

Übung

Ü. 5. Es soll geprüft werden, ob die Kundennummer 10 000 vorliegt. Die Kundennummern werden im Feld KNR im Zeichenformat zur Prüfung bereitgestellt.

6. Dezimalarithmetik

Die Befehle der Dezimalarithmetik verarbeiten Daten im gepackten Format. Sämtliche Befehle dieser Arithmetik sind Zweiadreßbefehle vom Typ SS-Format arithmetisch (Abschn. 2.6.). Die Verarbeitung der Operanden erfolgt während der Befehlsabarbeitung von rechts nach links. Durch die angegebenen Adressen wird jeweils das linke Byte des Operanden adressiert. Jeder Operandenadresse ist eine Operandenlänge zugeordnet. Sie gibt die Anzahl der Bytes an, die in die Operation einzubeziehen sind. Die Operandenfelder für die Dezimalarithmetik haben eine variable Länge zwischen 1 und 16 Bytes. Die Ergebnisse werden immer im Feld des ersten Operanden gespeichert. Die Vorzeichen der Ergebnisse von Dezimaloperationen sind intern wie folgt verschlüsselt:

positiv	(plus)	1100 $\triangleq$ X'C'
negativ	(minus)	1101 $\triangleq$ X'D'

Die Daten werden als rechtsbündig stehende ganze Zahlen aufgefaßt.
Der Befehlssatz der Dezimalarithmetik umfaßt Befehle zur Ausführung von Addition, Subtraktion, Multiplikation, Division und zur Ausführung von Vergleichsoperationen.
Da die Daten allgemein im erweiterten (ungepackten) Format ein- bzw. ausgegeben werden, die Dezimalarithmetik jedoch gepacktes Operandenformat verlangt, sind Befehle zur Umwandlung der beiden Datenformate vorhanden (Packen, Entpacken).

6.1. Packen und Entpacken

Der typische Ablauf bei Anwendung der Dezimalarithmetik ist aus Bild 14 ersichtlich.

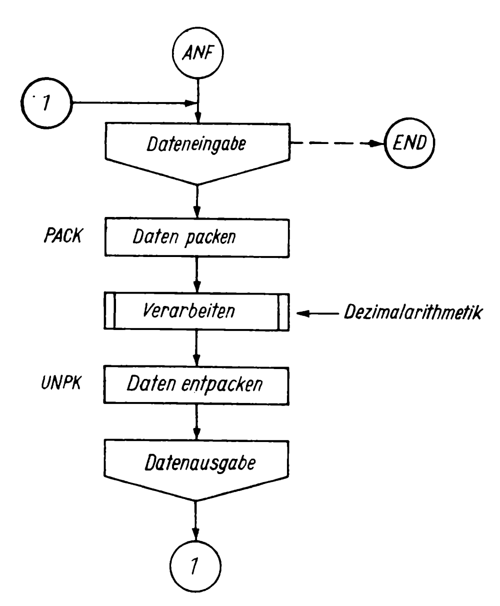
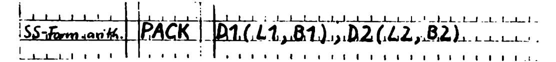


Bild 14. Typischer Ablauf bei Anwendung der Dezimalarithmetik

6.1.1. *Packen*



Befehlswirkung. Der ungepackte Inhalt des durch D2(B2) adressierten Feldes wird in der Länge von L1 gepackt und im Feld des ersten Operanden, Adresse D1(B1), gespeichert.

Bemerkung

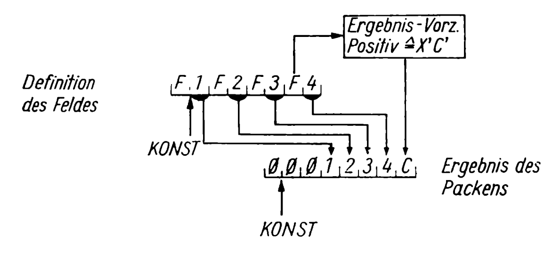
1. Die Feldlängen müssen nicht übereinstimmen. Ist Operand 1 zu lang, so werden in die linken Stellen des ersten Operanden binäre Nullen (X'00') eingesteuert. Wenn Operand 1 zu kurz ist, werden die restlichen (linken) Ziffern des zweiten Operanden nicht übertragen.
2. Zum Ablauf beim Packen:
   Austausch des Ziffern- und Zonenteils des rechten Bytes im Quellfeld bei der Übertragung in das Zielfeld (Ergebnisvorzeichen eingesteuert, s. Beispiel);
   danach werden aus dem Quellfeld nach links fortschreitend nur die Ziffernteile (numerische Teile) der Bytes in das Zielfeld übertragen; dabei werden zwei Ziffern in einem Byte abgespeichert (gepackt).

Beispiel

Es soll in das gleiche Feld KONST gepackt werden:

```
         PACK  KONST,KONST
          :
KONST    DC    C'1234'
```

Das Ergebnis der Operation sieht wie folgt aus (hexadezimale Darstellung):



6.1.2. *Entpacken*

SS-Form.arik.	UNPK	D1(L1,B1),D2(L2,B2)

Befehlswirkung. Der gepackte Inhalt des durch D2(B2) adressierten Feldes wird in der Länge von L1 entpackt und im Feld des ersten Operanden. Adresse D1(B1), gespeichert.

Bemerkung

1. Die Feldlängen müssen nicht übereinstimmen. Ist der erste Operand zu lang, werden in die linken Stellen des ersten Operanden dezimale Nullen (X'F0') eingefügt. Ist der erste Operand zu kurz, werden die restlichen (linken) Ziffern des zweiten Operanden nicht übertragen.

2. Zum Ablauf beim Entpacken:

   Austausch des Ziffern- und Zonenteils des rechten Bytes im Quellfeld bei der Übertragung in das Zielfeld;

   danach wird jeder Ziffer des Quellfelds, nach links fortschreitend, bei der Übertragung in das Zielfeld ein Zonenteil (X'F') zugefügt.

3. Im allgemeinen führt ein Entpacken in das gleiche Feld zu einem Überschreiben gültiger Informationen und sollte aus diesem Grund vermieden werden.

4. Bei einer Ausgabe (z. B. über Drucker) von Werten, die durch UNPK entpackt wurden, muß beachtet werden, daß das Ergebnisvorzeichen noch vorhanden ist. Dieses Vorzeichen (positiver Wert ≙ X'C') ist durch das entsprechende Vorzeichen des Maschinenschlüssels (positiver Wert ≙ X'F') zu ersetzen (Befehl MVZ).

Hinweis. Eine andere Art der Vorzeichensteuerung wird im Abschn. 12. behandelt.

Übung

Ü. 6. Der Inhalt des Feldes BER1 ist in das Feld BER2 zu entpacken. BER2 soll anschließend über Drucker ausgegeben werden.

6.2. Operationen mit Dezimalzahlen

6.2.1. *Grundrechenarten*

6.2.1.1. *Addition*

```
SS-Form.arith. | AP    | D1(L1,B1),D2(L2,B2)
```

Befehlswirkung

⟨D1(B1)⟩ := ⟨D1(B1)⟩ + ⟨D2(B2)⟩

Bemerkung

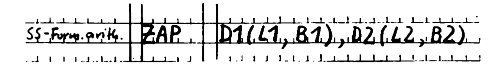
1. Die Operanden müssen gepackte Dezimalzahlen sein, sonst Programmausnahmeunterbrechung.
2. Die Addition erfolgt vorzeichenrichtig.
3. Bei ungleichen Feldlängen wird das kürzere Feld so behandelt, als ob es führende Nullen hätte.
4. Der Bedingungskode (Abschn. 7.1.) wird gestellt.

Beispiel

Zum Inhalt des Feldes SUM (gepackte Zahl) soll die Konstante „1000“ addiert werden. Dabei wird günstig mit einem Literal gearbeitet.

```
        | AP | SUM,=P'1ØØØ'          <SUM> vor Addition
        | :  |                        Ø1 23 45 6C
SUM     | DC | P'123456'
oder mit Konstantendefinition:        <SUM> nach Addition
        | AP | SUM,KONST              Ø1 24 45 6C
        | :  |
SUM     | DC | P'123456'
KONST   | DC | P'1ØØØ'
```

6.2.1.2. *Löschen und Addieren*



Befehlswirkung. Das durch D1(B1) adressierte Feld wird auf Null (P'0') gelöscht. Zu diesem Nullfeld wird der Inhalt des durch D2(B2) adressierten Feldes addiert.

Bemerkung

1. Der Inhalt des durch D2(B2) adressierten Feldes muß eine gepackte Dezimalzahl sein.
2. Bei ungleichen Feldlängen wird das kürzere Feld so behandelt, als ob es führende Nullen hätte.
3. Der Bedingungskode wird gestellt.

4. ZAP wird vorwiegend angewendet, um eine gepackte Dezimalzahl in ein Feld bestimmter Länge zu übertragen. Dieser Fall tritt bei Datenkonvertierungen (Abschn. 8.1.) auf.

Beispiel

Das Feld SUMME soll auf Null (P'0') gelöscht werden.

	ZAP	SUMME,=P'0' Wirkung: 0000000000C
SUMME	DS	CL5 SUMME

6.2.1.3. *Subtraktion*

SS-Form.orik.	SP	D1(L1,B1),D2(L2,B2)

Befehlswirkung

⟨D1(B1)⟩ := ⟨D1(B1)⟩ − ⟨D2(B2)⟩

Bemerkung. Vgl. Addition.

6.2.1.4. *Multiplikation*

SS-Form.orik.	MP	D1(L1,B1),D2(L2,B2)

Befehlswirkung

⟨D1(B1)⟩ := ⟨D1(B1)⟩ · ⟨D2(B2)⟩

Bemerkung

1. Der Multiplikand und der Multiplikator müssen gepackte Dezimalzahlen sein.

2. Das Produkt wird im Feld des ersten Operanden (Produktfeld) gebildet.

3. Vor der Multiplikation muß der Multiplikand rechtsbündig mit führenden Nullen im Produktfeld stehen (dafür Befehle: PACK, ZAP).

4. Die Länge des Produktfelds in Bytes ergibt sich aus der Addition der Längen des gepackten Multiplikanden[Bytes] und des gepackten Multiplikators [Bytes]. |

5. Die Länge des Multiplikators darf maximal 8 Bytes (15 Ziffern + VZ) betragen.

6. Der Bedingungskode wird nicht gestellt.
Bei der Programmierung der Multiplikation sind die im Bild 15 dargestellten Überlegungen zu treffen.

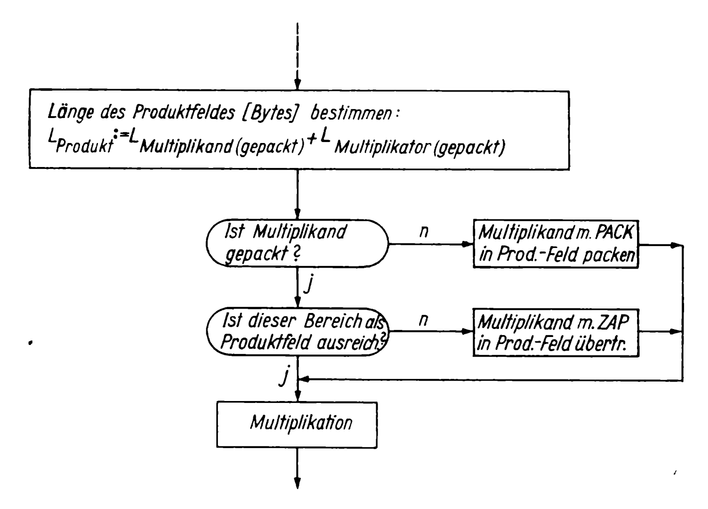


Bild 15. Überlegungen bei der Programmierung der Multiplikation

Beispiel

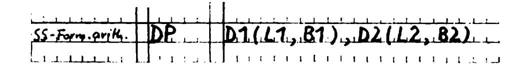
Der Inhalt des Feldes WERT ist mit dem Faktor 25 zu multiplizieren.

```
        ZAP   PROD,WERT
        MP    PROD,FAKTOR
          :
WERT    DC    P'400'
FAKTOR  DC    P'25'
PROD    DS    CL4
```

```
40 0C      02 5C
↑WERT      ↑FAKTOR

00 10 00 0C
↑PROD
```

6.2.1.5. *Division*



Befehlswirkung

⟨D1(B1)⟩ := ⟨D1(B1)⟩ : ⟨D2(B2)⟩

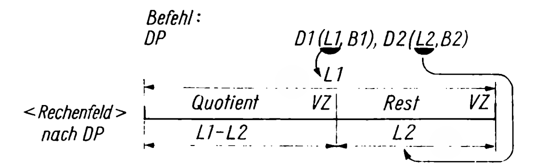
Bemerkung

1. Dividend und Divisor müssen gepackte Dezimalzahlen sein.

2. Vor der Division muß der Dividend rechtsbündig mit führenden Nullen im Feld des ersten Operanden (Rechenfeld) stehen (dafür Befehle: PACK, ZAP).

3. Die Länge des Rechenfelds in Bytes ergibt sich aus der Addition der Längen des gepackten Dividenden [Bytes] und des gepackten Divisors [Bytes].

4. Nach der Division steht in den linken Stellen des Rechenfelds der Quotient in der Länge L1 — L2, der Rest in den rechten Stellen in der Länge L2.



5. Das Vorzeichen des Quotienten wird nach algebraischen Regeln bestimmt. Das Vorzeichen des Restes entspricht dem des Dividenden.

6. Der Bedingungskode wird nicht gestellt.

6.2.2. *Verschieben und Runden*

Ein Verschieben und Runden wird in Verbindung mit den Befehlen Multiplikation und Division benötigt.

Es liege folgende Problematik vor: Ein Wert mit zwei Kommastellen soll mit einem Faktor (drei Kommastellen) multipliziert werden. Das Ergebnis soll auf zwei Kommastellen berechnet werden.

x x x x x C * Ø y y y y C

Multiplikand *Multiplikator*

Produktstruktur

⟹ Ø Ø a b c d e f g h i C

↑PROD

1. RUNDEN

g = zu rundende Ziffer; Runden durch Addition mit Rundungskonstante 5 Ø Ø C

wenn g { *<5, kein Übertrag auf f* ; *≧5, Übertrag auf f* }

2. VERSCHIEBEN nach rechts

g, h, i sollen nach rechts aus dem Feld heraus-geschoben werden. Das Vorzeichen muß erhalten bleiben

Somit:

Ø Ø Ø Ø Ø a b c d e f C

↑PROD

Entsprechender Befehl: MVO PROD, PROD(4)

(vgl. Abschn. 6.2.2.2.)

6.2.2.1. *Runden*

Im allgemeinen ist nicht bekannt, ob die zu rundende Zahl positiv oder negativ ist. Es wäre also falsch, immer mit einer positiven Rundungskonstanten zu arbeiten. So ist folgender Ablauf zweckmäßig:

1. Übertragen des Vorzeichens des zu rundenden Wertes auf die Rundungskonstante (Befehl: MVN).
2. Addition der nunmehr mit dem richtigen Vorzeichen behafteten Rundungskonstanten zu dem zu rundenden Wert.

Beispiel

Runden eines Zahlenwerts mit unbestimmtem Vorzeichen:

Ø 4 3 6 8 1 5 VZ
FELD — zu rundende Ziffer

Rundungskonstante 5 Ø Ø C
KONST

gerundeter Wert Ø 4 3 7 3 1 5 VZ
FELD

Die Befehlsfolge lautet:

	MVN	KONST+1(1),FELD+3
	AP	FELD,KONST
	:	
KONST	DC	P'5ØØ'
FELD	DS	CL4

6.2.2.2. *Verschieben*

Linksverschieben

Vor der Division ist es oft notwendig, die Zahl der Dezimalstellen von Dividend und Divisor anzugleichen.

Beispiel

x x x ▽ C — Dividend

▽ y y y C — Divisor

Vor Division muß Dividend um zwei Stellen nach links versetzt werden ↗ Multiplikation mit Faktor 100

Rechtsverschieben

Für die im Abschn. 6.2.2. dargelegte Problematik ist der Befehl „Transport mit Absetzen" (MVO) anwendbar.

SS-Form prikt.	MVO	D1(L1,B1),D2(L2,B2)

Befehlswirkung. Der Inhalt des durch D2(B2) adressierten Feldes wird um ein Halbbyte versetzt in das durch D1(B1) adressierte Feld übertragen. Der Ziffernteil des am weitesten rechts stehenden Bytes des ersten Operanden wird nicht in die Operation einbezogen. Dieses Halbbyte bleibt somit unverändert erhalten.

Bemerkung

1. MVO ist ein Befehl im arithmetischen SS-Format. Er arbeitet also von rechts nach links.

2. Ist der erste Operand zu lang, werden in die linken Bytes binäre Nullen (X'00') eingesteuert. Ist der erste Operand zu kurz, werden die linken Bytes des zweiten Operanden nicht übertragen.

3. Der Bedingungskode wird nicht gestellt.

4. Eine Anwendung wurde im Abschn. 6.2.2. gezeigt. Beim Verschieben einer geraden Anzahl Ziffern nach rechts kann der Befehl MVO zweimal angewendet werden.

6.2.3. *Vergleichen*

SS-Form.-arith.	CP	D1(L1,B1),D2(L2,B2)

Befehlswirkung. Es wird getestet, ob die Differenz erster Operand minus zweiter Operand kleiner, gleich oder größer Null ist.

Bemerkung

1. Die Operanden müssen gepackte Dezimalzahlen sein.

2. Der Vergleich erfolgt byteweise von rechts nach links in der Länge des längeren Operanden. Der kürzere Operand wird so behandelt, als ob er führende binäre Nullen hätte.

3. Das Vergleichsergebnis wird im Bedingungskode festgehalten und bezieht sich auf die Relation erster zu zweitem Operanden.

7. Programmverzweigungen

Verzweigungen im Programm ermöglichen es, die sequentielle Befehlsabarbeitung zu unterbrechen und das Programm an einer anderen Stelle fortzusetzen. Die Operationen zur Programmverzweigung können in *unbedingte* (Verzweigen in jedem Fall) und *bedingte Verzweigungsoperationen* (Verzweigen nur bei erfüllter Verzweigungsbedingung) unterschieden werden. Es soll beispielsweise zu einer Adresse ADR verzweigt werden, wenn der Inhalt des Feldes OP1 größer als der Inhalt des Feldes OP2 ist. Ein Vergleichsbefehl stellt fest, ob die Bedingung „⟨OP1⟩ größer ⟨OP2⟩" erfüllt ist. Die durch den Befehl erzielte Vergleichsaussage wird in den Bitpositionen 34 und 35 des PSW (Abschn. 1.1.2.1.) gespeichert. Dieses Festhalten der Vergleichsaussage wird als *Stellen des Bedingungskodes* bezeichnet. Die Stellung des Bedingungskodes wird im Programm ausgewertet.

7.1. Bedingungskode

Das Ergebnis von einigen Operationen (z. B. arithmetische Operationen, Vergleichsoperationen) wird zum Stellen des Bedingungskodes verwendet. Dieses Stellen des Bedingungskodes ist mit dem Anzeigen eines bestimmten Zustands vergleichbar. So wird beispielsweise angezeigt, ob das Ergebnis einer Addition kleiner, gleich oder größer als Null ist, oder eine erzielte Vergleichsaussage wird gespeichert usw. Für den Bedingungskode stehen im PSW die Bitpositionen 34 und 35 zur Verfügung.
Auf Grund der zwei Bitpositionen ergeben sich für den Bedingungskode (BC) vier Belegungsmöglichkeiten: (0, 1, 2, 3).
Wesentlich ist, daß ein Auswerten des Bedingungskodes *vor* einer Operation zu erfolgen hat, die den Bedingungskode erneut beeinflußt. Diese Auswertung erfolgt durch die Verzweigungsbedingung.

7.2. Verzweigungsbedingung

Die Verzweigungsbedingung ist Bestandteil des Befehls, der den Stand des Bedingungskodes auswerten soll.
Sie wird als eine 4 bit umfassende Maske im Befehl angegeben. Die Zuordnung von BC und Verzweigungsbedingung ist wie folgt getroffen:

Bedingungskode	Verzweigungsbedingung (Maske)	Bei Vergleich	Bei arithmetischen Operationen
00 = 0	1000 = 8	gleich	= 0
01 = 1	0100 = 4	kleiner	< 0
10 = 2	0010 = 2	größer	> 0
11 = 3	0001 = 1		Überlauf

Sonderfälle

0000 (keine Operation)
1111 (unbedingte Verzweigung)

Die genannten Verzweigungsbedingungen können beliebig kombiniert werden (0 bis 15).
Jedem der vier genannten Bedingungskodes ist eine Bitposition der Maske zugeordnet. Es findet dann eine Programmverzweigung statt, wenn dem zum Zeitpunkt der Befehlsausführung aktuellen BC in der zugeordneten Bitposition der Maske ein mit „1“ belegtes Bit gegenübersteht.

7.3. Befehle

7.3.1. *Verzweigen nach Bedingung*

Die Befehle können im RR-Format und RX-Format geschrieben werden:

RR-Format	BCR	M1,R2
		→ <R2> = Verzweigungsadresse
RX-Format	BC	M1,D2(X2,B2)
		<X2> + <B2> + D2 = Verzweigungsadresse

Befehlswirkung. Das Programm verzweigt zu der im zweiten Operanden angegebenen Adresse, wenn die Verzweigungsbedingung erfüllt ist (Abschnitt 7.2.).

Bemerkung

1. Bei BCR muß die Verzweigungsadresse im adressierten Register R2 stehen.

2. Ist M1 = 0000, wird nicht verzweigt (keine Operation).

3. Ist M1 = 1111, wird unbedingt verzweigt.

4. Ist die Verzweigungsbedingung nicht erfüllt, wird in der sequentiellen Befehlsfolge fortgesetzt.

Beispiel

Es soll zur Adresse ADR verzweigt werden, wenn ⟨OP1⟩ größer als ⟨OP2⟩ ist.

```
         CLC   OP1,OP2
         BC    2,ADR
          :
ADR       :
```

7.3.2. *Erweiterter Operationskode*

Im Abschn. 7.3.1. wurde gezeigt, daß die Verzweigungsbedingung im Operandenfeld mit anzugeben war. Die Assemblersprache stellt hier eine Vereinfachung bereit. Anstelle der BC-Befehle können erweiterte Operationskodes verwendet werden. Diese Operationskodes geben außer der Anweisung zum Verzweigen auch die Bedingung an, unter der verzweigt werden soll. In Tafel 2 sind diese Operationskodes zusammengestellt [5, S. 28].

Damit lautet die Befehlsfolge für Abschn. 7.3.1. wie folgt:

```
         CLC   OP1,OP2
         BH    ADR
          :
ADR
```

7.3.3. *Verzweigen und Laden Folgeadresse*

Es liegt folgende Problematik vor: Das Programm soll an einer bestimmten Stelle verzweigt werden (z. B. Absprung in ein Unterprogramm). Diese Verzweigung kann durch einen Sprungbefehl erreicht werden. Dabei wird gefordert, daß die Adresse des auf den Sprungbefehl folgenden Befehls für weitere Auswertungen gespeichert wird. Diese Adresse soll mit *Folgeadresse* bezeichnet werden.

Tafel 2. Erweiterte Operationskodes

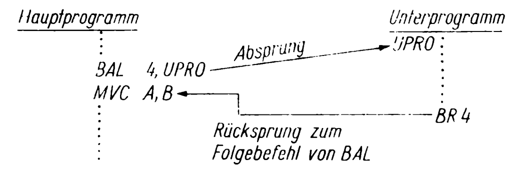
Erweiterter Operationskode		Bedeutung	Entsprechender Befehl	
B	D2(X2,B2)	unbedingter Sprung	BC	15,D2(X2,B2)
BR	R2	unbedingter Sprung	BCR	15,R2
NOP	D2(X2,B2)	kein Sprung	BC	0,D2(X2,B2)
NOPR	R2	kein Sprung	BCR	0,R2
Verzweigen nach Vergleichsoperationen				
BH	D2(X2,B2)	Sprung bei größer	BC	2,D2(X2,B2)
BL	D2(X2,B2)	Sprung bei kleiner	BC	4,D2(X2,B2)
BE	D2(X2,B2)	Sprung bei gleich	BC	8,D2(X2,B2)
BNH	D2(X2,B2)	Sprung bei nicht größer	BC	13,D2(X2,B2)
BNL	D2(X2,B2)	Sprung bei nicht kleiner	BC	11,D2(X2,B2)
BNE	D2(X2,B2)	Sprung bei nicht gleich	BC	7,D2(X2,B2)
Verzweigen nach arithmetischen Operationen				
BO	D2(X2,B2)	Sprung bei Überlauf	BC	1,D2(X2,B2)
BP	D2(X2,B2)	Sprung bei positiv	BC	2,D2(X2,B2)
BM	D2(X2,B2)	Sprung bei negativ	BC	4,D2(X2,B2)
BZ	D2(X2,B2)	Sprung bei Null	BC	8,D2(X2,B2)
BNP	D2(X2,B2)	Sprung bei nicht positiv	BC	13,D2(X2,B2)
BNM	D2(X2,B2)	Sprung bei nicht negativ	BC	11,D2(X2,B2)
BNZ	D2(X2,B2)	Sprung bei nicht Null	BC	7,D2(X2,B2)
Verzweigen nach „Prüfen unter Maske“ (Abschn. 10.2.)				
BO	D2(X2,B2)	Sprung bei BC 3	BC	1,D2(X2,B2)
BM	D2(X2,B2)	Sprung bei BC 1	BC	4,D2(X2,B2)
BZ	D2(X2,B2)	Sprung bei BC 0	BC	8,D2(X2,B2)
BNO	D2(X2,B2)	Sprung bei BC 0,1	BC	14,D2(X2,B2)

Für die geforderte Programmverzweigung einschließlich des Ladens der Folgeadresse stehen Befehle im RR-Format und im RX-Format zur Verfügung:

RR-Format	BALR	R1,R2
RX-Format	BAL	R1,D2(X2,B2)

Befehlswirkung. Die Adresse des Folgebefehls wird nach R1 geladen. Außerdem wird zur Adresse D2(X2,B2) bzw. zu der im Register R2 stehenden Adresse verzweigt.

Anwendung. Es soll in ein Unterprogramm mit Namen UPRO verzweigt werden. Die Rückkehradresse sei im Register 4 zu speichern:



Besonderheiten bei BALR. Ist R2 = 0, findet keine Verzweigung statt. Es wird nur die Adresse des auf BALR folgenden Befehls in das Register R1 geladen. Diese Variante wird verwendet, um Basisregister mit der Speicheradresse des Folgebefehls zu laden.

Beispiel

	START	
ANF	BALR	4,Ø
	USING	*,4
	MVC	BER,KNR
	:	
BER	DS	CL...
KNR	DS	CL...
	END	ANF

Die Assembleranweisung USING (Abschn. 4.3.3.) teilt dem Assembler zum Zeitpunkt der Übersetzung mit, welches Register als Basisregister verwendet werden soll (hier: Register 4). Außerdem wird dem Assembler durch USING die Basisadresse (hier: Adresse des MVC-Befehls) übermittelt. Zum Zeitpunkt der Programmabarbeitung wird im Beispiel durch BALR das Register 4 mit der Adresse des MVC-Befehls geladen. Es wird nicht verzweigt.

8. Binärarithmetik

Neben der im Abschn. 6. beschriebenen Dezimalarithmetik kann mit der Binärarithmetik gerechnet werden. Die Anwendung der Binärarithmetik setzt voraus, daß Binärdaten fester Länge (Wort- bzw. Halbwortformat, Abschn. 2.5.) vorliegen.
Vom Standpunkt der Ein- und Ausgabe ist das ungepackte Datenformat als das „natürliche" Format anzusehen. Für die Anwendung der Binärarithmetik müssen die Operanden in das entsprechende Datenformat *konvertiert* werden. Diese Konvertierungen, also das Umwandeln von Dezimalzahlen in Dualzahlen (und umgekehrt), sollen nachstehend behandelt werden. Bild 16 gibt einen Überblick über die Möglichkeiten der Konvertierung.

8.1. Datenkonvertierungen

8.1.1. *Umwandeln Dezimalzahl in Dualzahl*

RX-Format	CVB	R1,D2(X2,B2)

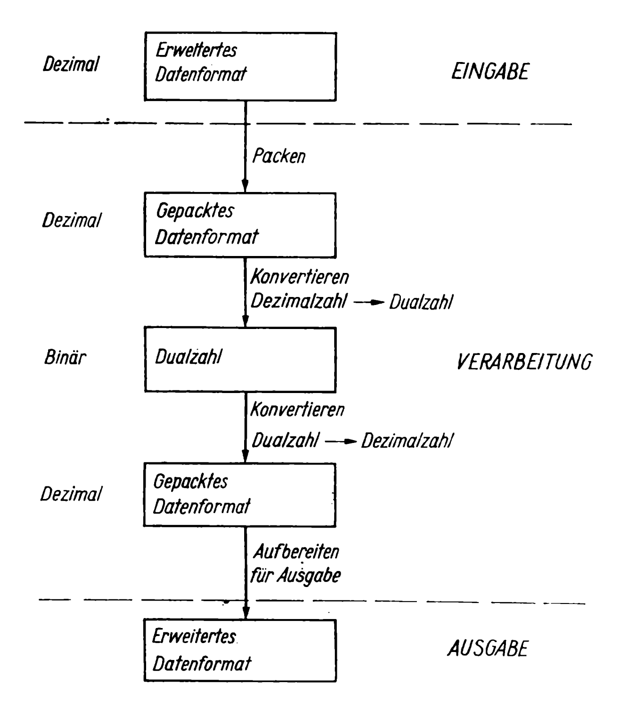


Bild 16. Möglichkeiten der Datenkonvertierung

Befehlswirkung. Die auf der Adresse D2(X2,B2) stehende gepackte Dezimalzahl wird vorzeichenrichtig in ihr Dualäquivalent umgewandelt. Diese Dualzahl wird in das Register R1 geladen.

Bemerkung

1. Die umzuwandelnde Dezimalzahl muß in gepacktem Format in der Länge von 8 Bytes auf der Adresse D2(X2,B2) gespeichert sein.

2. Die Speicheradresse D2(X2,B2) muß an einer Doppelwortgrenze beginnen (durch 8 teilbar sein).

3. Die Zahl z wird sowohl vor der Umwandlung als auch danach als rechtsbündig interpretiert.

4. Auf Grund der Registerkapazität (32 bit) ergibt sich für die Größe der umzuwandelnden Zahl z folgende Einschränkung:

$$-2^{31} \leqq z \leqq 2^{31} - 1$$

bzw.

$$-2\,147\,483\,648 \leqq z \leqq 2\,147\,483\,647$$

Anwendung. Die auf der Adresse ZAHL stehende 4 Bytes lange gepackte Dezimalzahl soll als Dualzahl im Register 5 bereitgestellt werden.

```
        ZAP  KONVERT,ZAHL    <ZAHL> → KONVERT
        CVB  5,KONVERT       <KONVERT>dezimal → Reg.5dual
        :
ZAHL    DS   CL4
KONVERT DS   D
```

8.1.2. *Umwandeln Dualzahl in Dezimalzahl*

```
RX-Format CVD  R1,D2(X2,B2)
```

Befehlswirkung. Die in Register R1 stehende Dualzahl wird vorzeichenrichtig in ihr Dezimaläquivalent umgewandelt und auf der Adresse D2(X2,B2) in der Länge von 8 Bytes abgespeichert.

Bemerkung. Die Speicheradresse D2(X2,B2) muß an einer Doppelwortgrenze beginnen.

8.2. Operationen mit Festkommadaten

Der Befehlssatz der Binärarithmetik mit Festkommadaten realisiert die Funktionen

Laden und Speichern
Verschieben
Durchführen der Grundrechenarten
Vergleichen.

Die Operationen der Binärarithmetik werden in Registern ausgeführt. Dazu ist notwendig, Register zu laden bzw. Registerinhalte abzuspeichern.

8.2.1. *Laden und Speichern*

Laden Adresse

```
RX-Format LA   R1,D2(X2,B2)
```

Befehlswirkung. Die Speicheradresse des zweiten Operanden wird in die rechten 24 Bitpositionen des Registers R1 eingetragen. Die ersten 8 Bitpositionen des Registers R1 werden mit Nullen aufgefüllt.

Bemerkung. Vor Ausführung des eigentlichen Ladens des Registers wird die effektive Speicheradresse wie folgt berechnet:

  Inhalt des allgemeinen Registers B2
+ Inhalt des Indexregisters X2
+ Verschiebung D2
―――――――――――――――――――――
= effektive HS-Adresse

Anwendungen

1. Nach Register 3 soll die Adresse von ZYKL geladen werden.

```
        LA    3,ZYKL(Ø)
                ⤷ D2(B2)  → X2
```

Die Adresse ZYKL wird vom Assembler in Basisregister B2 und Verschiebung D2 zerlegt. Das Register X2 wird hier nicht verwendet.

Bemerkung. Ist für Registerangaben die Zahl Null angegeben, so ist damit nicht das Register 0 gemeint, sondern der Zahlenwert Null. Damit kann vereinfacht wie folgt geschrieben werden:

```
        LA    3,ZYKL
```

2. Die Dualzahl 100 soll in das Register 5 geladen werden.

```
        LA    5,1ØØ(Ø,Ø)
  oder
        LA    5,1ØØ
```

3. Der Inhalt des Registers 5 soll um den Wert 10 erhöht werden. Dabei sind folgende Schreibweisen bezüglich der Wirkung äquivalent:

```
        LA    5,1Ø(5,Ø)
  oder
        LA    5,1Ø(Ø,5)
  oder
        LA    5,1Ø(,5)
  oder
        LA    5,1Ø(5)
```

Bemerkung

1. Zeile. B2 = 0, d. h., effektive Adresse ergibt sich aus ⟨X2⟩ plus 10. Für X2 ist Register 5 angegeben.

2. Zeile. X2 = 0, effektive Adresse ergibt sich aus ⟨B2⟩ plus 10. Für B2 ist Register 5 angegeben.

3. Zeile. Das Komma steht als Auslassungszeichen für eine Belegung von X2. Darauf kann auch verzichtet werden *(4. Zeile)*. Für alle Varianten gilt, daß die gebildete effektive Adresse nach Register 5 geladen wird.

Laden Wort

```
RX-Format  L     R1,D2(X2,B2)
```

Befehlswirkung. Der Inhalt der Adresse D2(X2,B2) wird in der Länge von 4 Bytes in das Register R1 geladen.

Bemerkung. Die Adresse D2(X2,B2) muß auf Wortgrenze (durch 4 teilbar) ausgerichtet sein.

Diese Forderung nach integralen Grenzen (Abschn. 2.3.) soll kurz mit
  Halbwortgrenze
  Wortgrenze
  Doppelwortgrenze
aufgezeigt werden.

Anwendung. Der Inhalt des Feldes ZAHL soll nach Register 2 geladen werden.

```
        L     2,ZAHL
        :
        :
ZAHL    DS    F
```

Laden Halbwort

```
RX-Format   LH    R1,D2(X2,B2)
```

Befehlswirkung. Der Inhalt der Adresse D2(X2,B2) wird in die rechten 16 Bitpositionen des Registers R1 übertragen. Die linken 16 Bitpositionen von R1 werden bei Ausführung der Operation mit Nullen aufgefüllt, wenn der Operand positiv ist; mit eins (1) aufgefüllt, wenn der Operand negativ ist.

Bemerkung. Halbwortgrenze für D2(X2,B2).

Einfügen Zeichen

```
RX-Format   IC    R1,D2(X2,B2)
```

Befehlswirkung. Von der Adresse D2(X2,B2) wird 1 Byte in die rechten 8 Bitpositionen des Registers R1 übertragen. Die linken 24 Bits von R1 bleiben unverändert erhalten.

Umladen von Registerinhalten

```
RR-Format   LR    R1,R2
```

Befehlswirkung. Der Inhalt des Registers R2 wird in das Register R1 übertragen.

Laden mehrfach

```
RS-Format   LM    R1,R3,D2(B2)
                  ↑  ↑  └→ Woher
                  Von Bis
```

Befehlswirkung. Der Inhalt der Adressen D2(B2), D2(B2)+4, D2(B2) +8 usw. (je 4 Bytes) wird in die Register R1, R1+1, ..., R3 übertragen. Die Adresse D2(B2) verweist auf das erste (linke) Wort des in die Register zu ladenden Hauptspeicherbereichs.

Die Übertragung ist beendet, wenn die angegebenen Register geladen sind. Die Register werden in aufsteigender Folge geladen. Ist R3 < R1, geht die Adressierung von Register 15 auf Register 0 über.

Bemerkung. Wortgrenze für D2(B2).

Anwendung. Ab Adresse BER ist je 1 Wort in die Register 6 bis 9 zu laden.

```
         LM    6,9,BER     <BER> → R6, <BER+4> → R7
                           <BER+8> → R8, <BER+12> → R9
BER      DS    4F
```

Speichern Wort

```
RX-Format   ST    R1,D2(X2,B2)
```

Befehlswirkung. Der Inhalt des Registers R1 wird auf der Adresse D2(X2,B2) abgespeichert.

Bemerkung. Wortgrenze für D2(X2,B2).

Speichern Halbwort

```
RX-Format   STH   R1,D2(X2,B2)
```

Befehlswirkung. Die rechten 16 bit des Inhalts von R1 werden auf der Adresse D2(X2,B2) abgespeichert.

Bemerkung. Halbwortgrenze für D2(X2,B2).

Speichern Zeichen

```
RX-Format   STC   R1,D2(X2,B2)
```

Befehlswirkung. Die rechten 8 bit des Inhalts von R1 werden auf der Adresse D2(X2,B2) abgespeichert.

Speichern mehrfach

```
RS-Format   STM   R1,R3,D2(B2)
                  ↑  ↑  └→ Wohin
                  Von Bis
```

Befehlswirkung. Der Inhalt der Register R1, R1+1, ..., R3 wird ab der Adresse D2(B2) abgespeichert. Die Operation ist beendet, wenn der Inhalt des Registers R3 übertragen wurde.

Übung

Ü. 8.1. Das dritte Byte des Feldes ADR soll nach Register 4 übertragen werden (Zeichen einfügen).

Ü. 8.2. Der Inhalt der Register 6 bis 9 ist ab der Adresse BER abzuspeichern.

8.2.2. *Verschiebeoperationen*

Unter *Stellenverschieben* soll verstanden werden, daß alle Bits eines Registers nach links bzw. nach rechts bewegt werden. Bei den entsprechenden Operationen ist zwischen *arithmetischen* und *logischen Verschiebeoperationen* zu unterscheiden.
Im Rahmen der zu behandelnden Binärarithmetik interessieren primär arithmetische Verschiebeoperationen. Es soll im folgenden auch nur auf diese Art der Verschiebung eingegangen werden.
Logische Verschiebeoperationen sind in [6] beschrieben.

Linksverschiebung arithmetisch

RS-Format	SLA	R1,D2(B2)
		gibt an, um wieviel Bits verschoben wird

Befehlswirkung. Der Inhalt des Registers R1 wird um die angegebene Anzahl Bits nach links verschoben. Das Vorzeichenbit wird nicht in die Verschiebeoperation einbezogen.

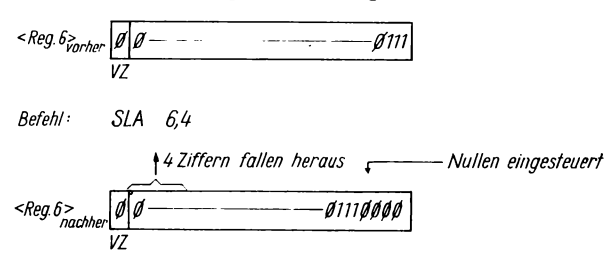
Bemerkung

1. Der Bedingungskode wird gestellt.

2. Eine Linksverschiebung um eine Bitposition entspricht einer Multiplikation des Operanden mit dem Faktor 2.

3. Fällt bei der Verschiebung eine dem Vorzeichen verschiedene Ziffer heraus, gibt es eine Festkomma-Überlaufunterbrechung.

Anwendung. Der Inhalt des Registers 6 soll mit dem Faktor 16 multipliziert werden. Eine Linksverschiebung von 4 bit realisiert diese Aufgabenstellung. Zum besseren Verständnis soll der Inhalt von Register 6 vor und nach der Operation dargestellt werden:



Rechtsverschiebung arithmetisch

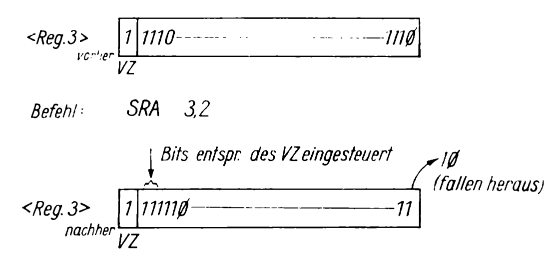
RS-Format	SRA	R1,D2(B2)

Befehlswirkung. Der Inhalt des Registers R1 wird mit Ausnahme des Vorzeichenbits um die angegebene Anzahl Bits nach rechts verschoben.

Bemerkung

1. Der Bedingungskode wird gestellt.
2. Eine Rechtsverschiebung um eine Dualstelle entspricht einer Division des Operanden mit dem Divisor 2.
3. Bei Rechtsverschiebung werden Bits entsprechend dem Vorzeichen eingesteuert.

Anwendung. Der Inhalt des Registers 3 soll um zwei Dualstellen nach rechts verschoben werden.



Die Operationen SLA und SRA beziehen sich auf ein Register. Im weiteren sollen Verschiebeoperationen behandelt werden, die im *doppelten Registerformat* arbeiten. Dabei wird ein Registerpaar gemeinsam verschoben. Unter *Registerpaar* oder *Doppelregister* soll die Zusammengehörigkeit zweier benachbarter allgemeiner Register R1 und R1+1 verstanden werden. So bilden beispielsweise die Register 4 und 5 ein Registerpaar. Das Vorzeichen steht im geradzahligen Register und gilt für das Registerpaar.

Linksverschiebung arithmetisch doppelt

RS-Format	SLDA	R1,D2(B2)
		geradzahlig

Befehlswirkung. Der Inhalt des adressierten Registerpaars [R1, R1+1] wird mit Ausnahme des Vorzeichenbits um die angegebene Anzahl Bitpositionen nach links verschoben. Es gelten die bei SLA aufgeführten Bemerkungen.

Bemerkung. R1 muß geradzahlig sein.

Rechtsverschiebung arithmetisch doppelt

RS-Format	SRDA	R1,D2(B2)
		geradzahlig

Befehlswirkung. Der Inhalt des adressierten Registerpaars [R1, R1+1] wird mit Ausnahme des Vorzeichenbits um die angegebene Anzahl Bits nach rechts verschoben. Es gelten die bei SRA aufgeführten Bemerkungen.

Bemerkung. R1 muß geradzahlig sein.
Eine Anwendung des Befehls SRDA wird bei der Division von Festkommazahlen (Abschn. 8.2.3.4.) behandelt.

8.2.3. *Grundrechenarten*

8.2.3.1. *Addition*

Addition Wort

RX-Format	A	R1,D2(X2,B2)

Befehlswirkung. $\langle R1 \rangle := \langle R1 \rangle + \langle D2(X2,B2) \rangle$

Bemerkung

1. Die Addition erfolgt vorzeichenrichtig.
2. Der Bedingungskode wird gestellt.
3. Wortgrenze für D2(X2,B2).

Addition Halbwort

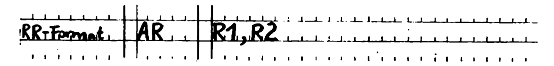
RX-Format	AH	R1,D2(X2,B2)

Befehlswirkung. $\langle R1 \rangle := \langle R1 \rangle + \langle D2(X2,B2) \rangle$

Bemerkung

1. Das auf der Adresse D2(X2,B2) stehende Halbwort wird vor der Addition durch Auffüllen mit dem Vorzeichenbit intern auf Wortformat ergänzt.
2. Der Bedingungskode wird gestellt.
3. Halbwortgrenze für D2(X2,B2).

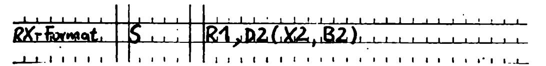
Addition von Registerinhalten



Befehlswirkung. $\langle R1 \rangle := \langle R1 \rangle + \langle R2 \rangle$
Bemerkung. Der Bedingungskode wird gestellt.

8.2.3.2. *Subtraktion*

Subtraktion Wort

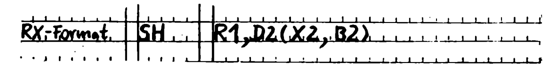


Befehlswirkung. $\langle R1 \rangle := \langle R1 \rangle - \langle D2(X2,B2) \rangle$

Bemerkung

1. Die Subtraktion erfolgt vorzeichenrichtig.
2. Der Bedingungskode wird gestellt.
3. Wortgrenze für D2(X2,B2).

Subtraktion Halbwort

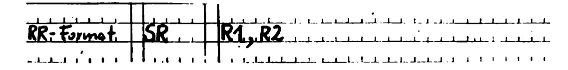


Befehlswirkung. ⟨R1⟩ := ⟨R1⟩ − ⟨D2(X2,B2)⟩

Bemerkung

1. Das auf der Adresse D2(X2,B2) stehende Halbwort wird vor der Subtraktion durch Auffüllen mit dem Vorzeichenbit intern auf Wortformat ergänzt.
2. Der Bedingungskode wird gestellt.
3. Halbwortgrenze für D2(X2,B2).

Subtraktion im Registerformat



Befehlswirkung. ⟨R1⟩ := ⟨R1⟩ − ⟨R2⟩

Bemerkung. Der Bedingungskode wird gestellt.

8.2.3.3. *Multiplikation*

Multiplikation im Wortformat

RX-Format. | M | R1,D2(X2,B2)

geradzahlig — Adresse des Multiplikators

Befehlswirkung. Der erste Operand wird vorzeichenrichtig mit dem zweiten Operanden multipliziert.

Bemerkung

1. Vor der Multiplikation muß der Multiplikand in das ungeradzahlige Rgister R1+1 des Registerpaars [R1, R1+1] geladen werden. Das Register R1 kann beliebigen Inhalt haben.
2. Der Multiplikator muß auf der Adresse D2(X2,B2) im Wortformat (an einer durch 4 teilbaren Adresse) abgespeichert sein.
3. Im Multiplikationsbefehl wird das geradzahlige Register R1 des Registerpaars [R1, R1+1] adressiert.
4. Das Produkt steht rechtsbündig im Registerpaar.
5. Der Bedingungskode wird nicht gestellt.

Multiplikation im Halbwortformat

RX-Format	MH	R1, D2(X2, B2)
		↑ beliebig — Adresse des Multiplikators

Befehlswirkung. Der erste Operand wird mit dem auf der Adresse D2(X2,B2) stehenden Halbwort vorzeichenrichtig multipliziert. Das Ergebnis wird nach Register R1 geladen.

Bemerkung

1. Halbwortgrenze für D2(X2,B2).
2. Der Bedingungskode wird nicht gestellt.
3. Das Ergebnis E ist nur dann richtig, wenn

$$-2^{31} \leqq E < 2^{31}$$

gilt.

4. Ein Überlauf wird nicht angezeigt. Wenn nicht klar erkennbar ist, daß das Ergebnis die unter 3. genannte Bedingung erfüllt, ist im Wortformat zu arbeiten.

Multiplikation im Registerformat

RR-Format	MR	R1, R2
		↑ geradzahlig

Befehlswirkung. Vgl. Multiplikation im Wortformat. Vor dem Befehl MR muß der Multiplikator in das Register R2 geladen werden.

8.2.3.4. *Division*

Division im Wortformat

RX-Format	D	R1, D2(X2, B2)
		↑ geradzahlig — Adresse des Divisors

Befehlswirkung. Der erste Operand wird vorzeichenrichtig durch den zweiten Operanden dividiert.

Bemerkung

1. Vor der Division muß der Dividend rechtsbündig im Registerpaar [R1, R1+1] stehen. Dabei wird günstig wie folgt programmiert:

   Laden des Dividenden nach R1

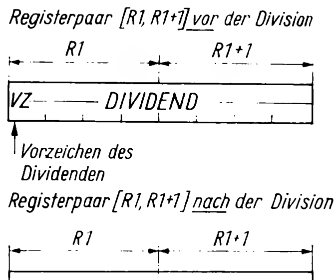
   Verschieben nach rechts (arithmetisch) um 32 bit (SRDA).

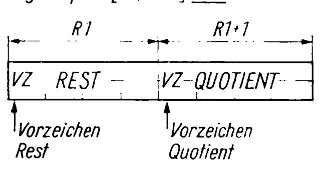
2. Der Divisor muß auf der Adresse D2(X2,B2) im Wortformat (an einer durch 4 teilbaren Adresse) abgespeichert sein.
3. Im Divisionsbefehl wird das geradzahlige Register R1 des Registerpaars [R1, R1+1] adressiert.

4. Der Quotient steht im ungeradzahligen Register R1+1. Das Vorzeichen des Quotienten wird algebraisch bestimmt.

5. Der Divisionsrest steht im geradzahligen Register R1. Das Vorzeichen des Restes entspricht dem des Dividenden.

Zum besseren Verständnis soll das Registerpaar [R1, R1+1] vor und nach der Division dargestellt werden:





Division im Registerformat

RR-Format	DR	R1, R2
		↑ geradzahlig

Befehlswirkung. Vgl. Division im Wortformat.
Der Divisor muß vor der Division in das Register R2 geladen werden.

8.2.4. *Vergleichsoperationen*

Der Vergleich von Festkommazahlen kann mit folgenden Befehlen realisiert werden:

1. Vergleich im Wortformat			
RX-Format	C	R1, D2(X2, B2)	Wortgrenze für D2(X2, B2)
2. Vergleich im Halbwortformat			
RX-Format	CH	R1, D2(X2, B2)	Halbwortgrenze für D2(X2, B2)
3. Vergleich im Registerformat			
RR-Format	CR	R1, R2	

Befehlswirkung. Es wird getestet, ob der Ausdruck „erster Operand minus zweiter Operand" kleiner, gleich oder größer Null ist. Das Ergebnis des Testes wird im Bedingungskode angezeigt. Diese Vergleichsaussage bezieht sich auf die Frage, wie sich der erste Operand zum zweiten Operanden verhält.

Bemerkung. Die Operanden werden durch den Vergleich nicht verändert.

Übung

Ü. 8.3. Es ist eine Division zu programmieren. Der Dividend stehe im Register 6. Als Divisor ist der Wert 10 zu verwenden. Der Quotient soll im Feld QUOT gespeichert werden. Ein möglicher Divisionsrest soll vernachlässigt werden.

9. Aufbau zyklischer Programme

Die Datenverarbeitung im technisch-ökonomischen Bereich ist primär dadurch gekennzeichnet, daß eine große Anzahl gleichartiger Daten nach der gleichen Rechenvorschrift nacheinander zu verarbeiten ist. Die Befehlsfolge zur Realisierung der jeweiligen Aufgabenstellung ist so aufzubauen, daß sie mehrfach (zyklisch) durchlaufen werden kann. Dabei muß gegeben sein, daß der Zyklus durch ein Endekriterium beendet wird.
So kommt es z. B. vor, daß eine Menge gleichartiger Daten gleichzeitig im HS steht (z. B. Tabellen) und mit der gleichen Befehlsfolge zu verarbeiten ist. Zur Adressierung dieser Daten müssen die entsprechenden Operandenadressen in den Befehlen variabel sein.
Diese Variabilität wird durch Verwendung von Registern erreicht. Dabei werden die einzelnen Hauptspeicheradressen der Operanden modifiziert (indiziert). Die Realisierung von Zyklen und die Möglichkeiten der Modifikation von Operandenadressen innerhalb von Zyklen sollen nachstehend behandelt werden. Methodisch wird dabei so vorgegangen, daß eine Aufgabe durch verschiedene Programmvarianten gelöst wird.

9.1. Zyklenzähler

Der *Zyklenzähler* zählt die Anzahl der Durchläufe durch einen Zyklus. Beim Erreichen einer bestimmten Anzahl von Durchläufen soll der Zyklus abgebrochen werden. Bei jedem Durchlauf soll der Zyklenzähler i um den Wert 1 verändert werden.
Die prinzipiellen Möglichkeiten der Realisierung von Zyklenzählern sind im Bild 17 dargestellt.
Für die programmtechnische Realisierung beispielsweise des Falles 2 bieten sich die Befehle BCT und BCTR an. Dabei umfaßt die Wirkung des jeweiligen Befehls die Programmschritte 3 und 4. Programmschritt 1 wird durch die Befehle „Laden" und „Laden Adresse" (Abschn. 8.2.1.) realisiert.

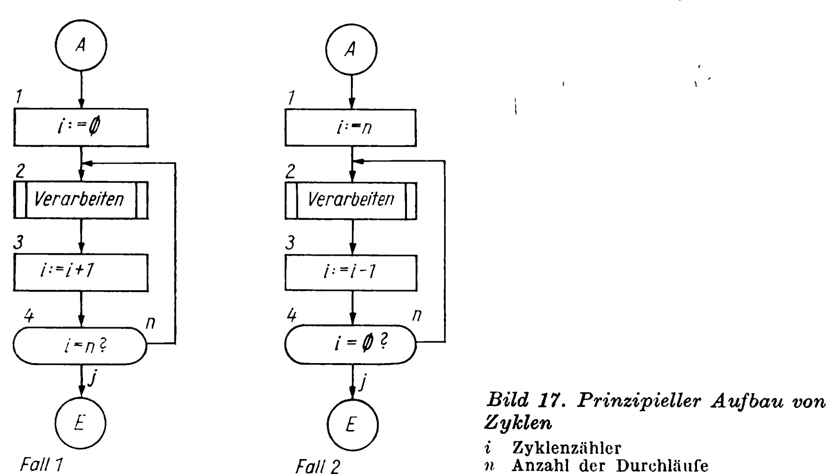


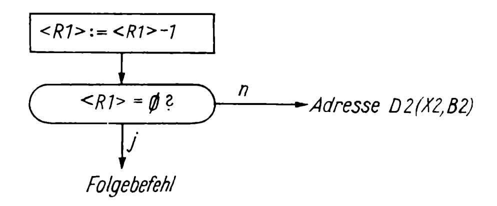
Bild 17. Prinzipieller Aufbau von Zyklen
i Zyklenzähler
n Anzahl der Durchläufe

9.1.1. *Verzweigen nach Zählwert*

Befehl BCT

RX-Format	BCT	R1,D2(X2,B2)

Befehlswirkung. Der Inhalt des Registers R1 wird um 1 vermindert. Falls der Inhalt von R1 dadurch zu Null geworden ist, wird das Programm bei der Folgeadresse fortgesetzt. Ist der neu gebildete Inhalt von R1 verschieden von Null, wird zur Adresse D2(X2,B2) verzweigt.



Befehl BCTR

RR-Format	BCTR	R1,R2

Befehlswirkung. Die Wirkung des Befehls BCTR entspricht der des Befehls BCT, jedoch steht die Verzweigungsadresse im Register R2.

Bemerkung. Ist für R2 der Wert Null angegeben (R2 = 0), wird nicht nach der im Register 0 stehenden Adresse verzweigt. Der Befehl vermindert dann nur den Zählwert, der als Dualzahl in R1 steht, um den Wert 1.

9.1.2. *Anwendungen*

Beispiel 1

Ein Zyklus soll 200mal durchlaufen werden. Die Zahl der Durchläufe soll im Register 4 gezählt werden.

Anwendung von	BCT:	
	L	4,=F'2∅∅'
ZYKL	:	
	:	
	BCT	4,ZYKL
WEITER	:	
Anwendung	von BCTR	
	LA	6,ZYKL
	L	4,=F'2∅∅'
ZYKL	:	
	:	
	BCTR	4,6
WEITER	:	

Beispiel 2

Auf einem Hauptspeicherbereich mit der Adresse ZAHL ist die Anzahl der Zyklendurchläufe im Wortformat (Dualzahl) bereitgestellt. Der Zyklenzähler soll durch Register 8 realisiert werden.

Name	Operation	Operanden
	L	8,ZAHL
ZYKL	:	
	.	
	BCT	8,ZYKL
WEITER	:	
	.	
ZAHL	DS	F

9.2. Befehlsmodifikationen im Zyklus

Im folgenden soll auf die *Verarbeitungsbefehle im Zyklus* eingegangen werden. Dabei wird davon ausgegangen, daß sich eine Menge von Daten (x_1, x_2, ..., x_n) im Speicher befindet, wobei bei jedem Zyklusdurchlauf auf ein anderes x_i ($i = 1, 2, \ldots, n$) durch einen Verarbeitungsbefehl zugegriffen wird.

Diese Variabilität der Operandenadressen wird dadurch erreicht, daß die im Befehl adressierten Register mit unterschiedlichem Inhalt versehen werden. Die Möglichkeit der Befehlsmodifikation im aufgezeigten Sinn ist bei Befehlen im RX-Format bzw. SS-Format (Abschn. 2.6.) gegeben.

Zum besseren Verständnis sind die *Möglichkeiten der Befehlsmodifikation* im Bild 18 dargestellt.

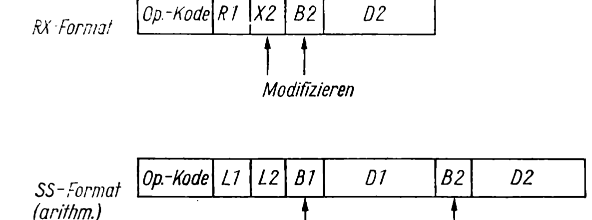


Bild 18. Möglichkeiten der Befehlsmodifikation im RX- bzw. SS-Format

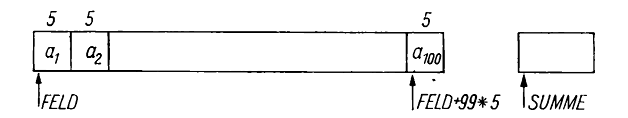
Im entsprechenden Befehl wird das zur Adressierung der Einzelwerte verwendete allgemeine Register anstelle des Basisregisters bzw. des Indexregisters verwendet. Dementsprechend wird unterschieden zwischen *Indizierung durch Basisregister* und *Indizierung durch Indexregister*. Die Variabilität der Operandenadressen kann bei Befehlen im SS-Format nur über Basisregisterindizierung, bei Befehlen im RX-Format über beide Arten der Indizierung erreicht werden (Bild 18).

9.2.1. *Indizieren durch Basisregister*

Die *Basisregisterindizierung* wird bei Befehlen im SS-Format angewendet. Die Problematik soll am folgenden Beispiel erläutert werden.

Beispiel

Im Hauptspeicher stehen auf dem Bereich FELD nach einer Dateneingabe 100 ungepackte Dezimalzahlen (a_1, a_2, ..., a_{100}) von je 5 Bytes Länge.



Es soll die Summe

$$S = a_1 + a_2 + \ldots + a_{100} = \sum_{i=1}^{100} a_i$$

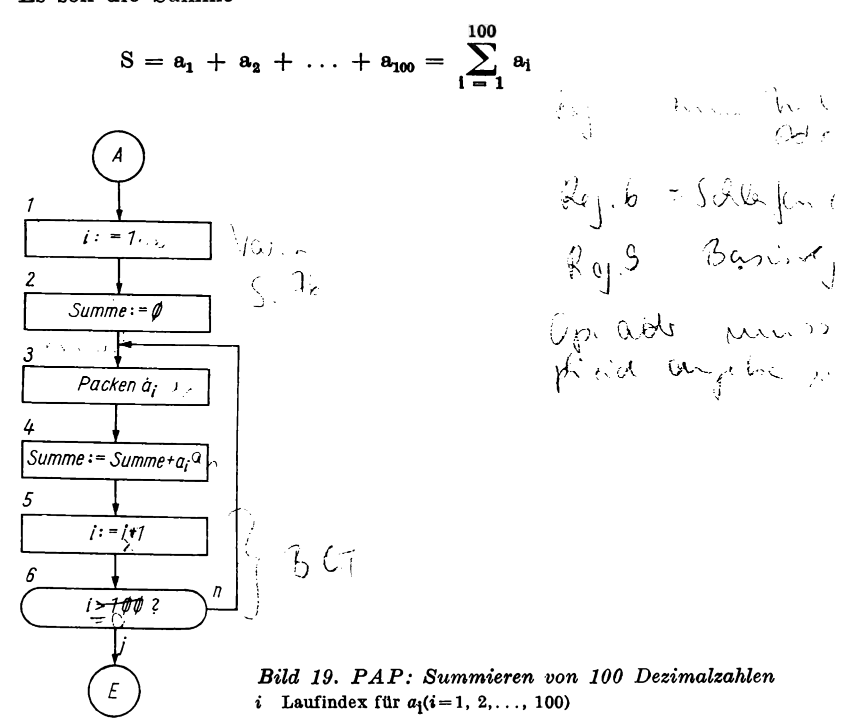


Bild 19. PAP: Summieren von 100 Dezimalzahlen
i Laufindex für a_i(i = 1, 2, ..., 100)

berechnet und im Feld SUMME in gepackter Form abgespeichert werden. Bild 19 zeigt den entsprechenden Programmablaufplan. Beim Aufbau des Programms könnte in folgenden Stufen vorgegangen werden:

1. Aufbau der Befehlsfolge für die eigentliche Verarbeitung (Packen, Addieren, Aktualisieren des Registerinhalts).

2. Aufbau der Befehlsfolge für den Zyklus mit den entsprechenden Ausgangsbedingungen für die in Punkt 1 genannte Befehlsfolge.

Stufe 1

```
        PACK   Ø(5,4),Ø(5,4)      Operandenlängen L1=L2=5 Bytes
        AP     SUMME,Ø(5,4)       S:=S+ai
        LA     4,5(4)             Erhöhen Registerinhalt
```

In den Programmschritten 4 und 5 wird auf die verschiedenen a_i zugegriffen. Die Operandenadressen müssen im Beispiel in der expliziten Form angegeben werden. Register 4 soll zum Indizieren verwendet werden. Damit beim ersten Durchlauf auch der entsprechende Wert a_1 in die Operation einbezogen wird, ist das Register 4 mit der Adresse FELD zu laden (s. Stufe 2). Die Werte a_i sollen in die gleichen Felder gepackt werden. Das Aktualisieren des Registerinhalts nach der Verarbeitung des jeweiligen Wertes erfolgt über „Laden Adresse".

Stufe 2

```
        LA     5,1ØØ              '1ØØ' → Reg. 5
        LA     4,FELD             >FELD< → Reg. 4
        ZAP    SUMME,=P'Ø'        SUMME löschen
ZYKL    }
        }      Verarbeitungsbefehle
        }
        BCT    5,ZYKL
```

Register 5 soll als Zyklenzähler in Verbindung mit dem Befehl „Verzweigen nach Zählwert" (BCT) fungieren.

Bemerkung. Das für den Programmteil verwendete Basisregister muß von dem zur Indizierung verwendeten verschieden sein.
Im Zusammenwirken ergibt sich folgende Befehlsfolge:

```
        BALR   9,Ø
        USING  *,9
        :      Dateneingabe
        .
        LA     5,1ØØ
        LA     4,FELD
        ZAP    SUMME,=P'Ø'
ZYKL    PACK   Ø(5,4),Ø(5,4)
        AP     SUMME,Ø(5,4)
        LA     4,5(4)
        BCT    5,ZYKL
        :
        :
FELD    DS     1ØØCL5
SUMME   DS     CL4
```

9.2.2. *Indizieren durch Indexregister*

Es soll von einem Beispiel analog zu Abschn. 9.2.1. ausgegangen werden.

Beispiel

Im Hauptspeicher stehen auf dem Bereich FELD 100 Festkommazahlen im Wortformat. Davon soll die Summe berechnet und im Feld SUMME gespeichert werden. Zum Summieren wird der Befehl „Addition (A)" (Abschn. 8.2.3.) verwendet.

Die Adressierung der Werte a_i erfolgt über ein Indexregister. Die Problematik besteht darin, daß im Indexregister immer die Entfernung des zu verarbeitenden Wertes a_i von der Anfangsadresse des Bereichs (hier: FELD) zu speichern ist.

Schematisch läßt sich dieser Sachverhalt wie folgt darstellen:

Durchlauf durch Zyklus	⟨Indexregister⟩	Adressierter Wert
1	0 * Schrittweite	a_1
2	1 * Schrittweite	a_2
3	2 * Schrittweite	a_3
.	.	.
.	.	.
.	.	.
100	99 * Schrittweite	a_{100}

Unter *Schrittweite* soll der konstante Abstand der zu verarbeitenden Werte a_i (hier: 4 Bytes) verstanden werden. Die Anfangsadresse des Bereichs FELD wird im Additionsbefehl in der symbolischen Form angegeben. Als Programmablaufplan für die vorliegende Aufgabenstellung kann der im Bild 19 dargestellte PAP verwendet werden.

```
         BALR   9,Ø
         USING  *,9
         :  Dateneingabe
         LA     5,1ØØ        Zyklenzähler einstellen
         LA     6,Ø          Indexreg. auf Ø * Schrittweite
         SR     7,7          R7 löschen
ZYKL     A      7,FELD(6)    S := S + a_i
         LA     6,4(6)       R6 um Schrittweite erhöhen
         BCT    5,ZYKL
         ST     7,SUMME      S abspeichern
         :
         :
FELD     DS     1ØØF
SUMME    DS     F
```

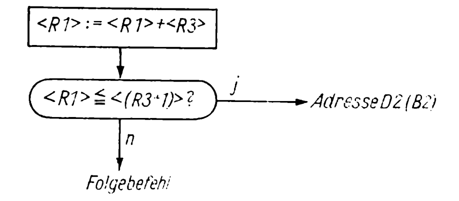
9.3. Indexorientierte Befehle in Zyklen

Im folgenden soll auf Möglichkeiten eingegangen werden, einen Zyklus auf Grund des Erreichens einer bestimmten Adresse (Endekriterium) abzubrechen. Dafür können die Befehle BXLE und BXH verwendet werden.

9.3.1. *Verzweigen bei Index „kleiner oder gleich“ (BXLE)*

RS-Format	BXLE	R1,R3,D2(B2)
		geradzahliges Register

Befehlswirkung. Die Inhalte der Register R1 und R3 werden addiert. Das Ergebnis der Addition wird nach Register R1 geladen. Dieser Inhalt des Registers R1 wird mit dem Registerinhalt (R3+1) verglichen. Wenn der Inhalt des Registers R1 kleiner oder gleich dem im Register (R3+1) gespeicherten Vergleichswert ist, wird zur Adresse D2(B2) verzweigt, ansonsten wird das Programm beim Folgebefehl fortgesetzt.



Bemerkung. Als Register R3 ist ein geradzahliges Register zu adressieren (z. B. Register 4). Damit steht das Register (R3+1) als das benachbarte Register bezüglich aufsteigender Registernumerierung fest (hier: Register 5). Der Befehl BXLE wird verwendet, um eine Verarbeitung von Werten in Richtung aufsteigender Operandenadressen zu organisieren. Die Register sind dabei mit folgenden Werten zu belegen:

R1	Anfangsadresse bzw. aktuelle Operandenadresse
R3	Schrittweite
R3+1	Endadresse.

Somit wird durch den Befehl BXLE

1. die Indizierung der Operandenadressen und
2. die Endeabfrage für den Zyklus

realisiert.

Beispiel

Die im Abschn. 9.2.1. behandelte Aufgabe soll mit dem Befehl BXLE gelöst werden. Es sollen folgende Register verwendet werden:

R1	Register 3 (Anfangsadresse: FELD)
R3	Register 4 (Schrittweite: 5)
R3+1	Register 5 (Endadresse: FELD + 99 ∗ 5).

Für das Laden der Register stehe folgende Befehlsfolge:

	LA	3,FELD	Anfangsadr.
	LA	4,5	Schrittweite
	LA	5,FELD+99*5	Endadr.

Anstelle der Befehlsfolge kann der Befehl „Laden mehrfach (LM)“ in Verbindung mit einer A-Adreßkonstanten (Abschn. 4.3.4.) angewendet werden.



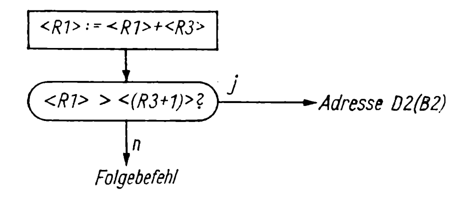
Damit ergibt sich folgende Befehlsfolge:

```
         LM    3,5,=A(FELD,5,FELD+99*5)   Register laden
         ZAP   SUMME,=P'0'                SUMME löschen
ZYKL     PACK  0(5,3),0(5,3)              Packen (gleiches Feld)
         AP    SUMME,0(5,3)               S := S + a_i
         BXLE  3,4,ZYKL                   R3 erhöhen
         :                                und Endeabfrage
FELD     DS    100CL5
SUMME    DS    CL4
```

9.3.2. *Verzweigen bei Index „größer“ (BXH)*

```
RS-Format   BXH   R1,R3,D2(B2)
                     ↑
                     geradzahliges Register
```

Befehlswirkung. Die Inhalte der Register R1 und R3 werden addiert. Das Ergebnis der Addition wird nach Register R1 geladen. Der Inhalt des Registers R1 wird mit dem Registerinhalt (R3+1) verglichen. Wenn der Inhalt des Registers R1 größer als der im Register (R3+1) gespeicherte Vergleichswert ist, wird zur Adresse D2(B2) verzweigt.



Bemerkung. Die Verarbeitung mehrerer im Hauptspeicher stehender Werte erfolgt bei Anwendung des Befehls BXH in Richtung absteigender Operandenadressen. Dabei ist in das Register R3 die Schrittweite als negativer Wert zu laden. In das Register (R3+1) ist die Anfangsadresse des zu verarbeitenden Bereichs, vermindert um die Schrittweite, zu laden.

Beispiel

Der Zyklus für das Summieren von 100 Zahlen (Abschn. 9.2.1.) soll mit dem Befehl BXH realisiert werden. Folgende Register werden verwendet:

R1	Register 5 (Adresse des ersten zu verarbeitenden Wertes: FELD + 99 $\ast$ 5)
R3	Register 6 (Schrittweite: −5)
R3+1	Register 7 (FELD−5)

Damit kann wie folgt programmiert werden:

```
         BALR   9,Ø
         USING  *,9
          :
         LA     3,FELD+99*5       Adr. erster zu verarb. Wert
         L      4,=H'-5'          Schrittweite
         LA     5,FELD-5          Anfangsadr. - Schrittweite
         ZAP    SUMME,=P'Ø'
ZYKL     PACK   Ø(5,3),Ø(5,3)
         AP     SUMME,Ø(5,3)
         BXH    3,4,ZYKL
WEITER    :
FELD     DS     1ØØCL5
SUMME    DS     CL4
```

10. Bitorientierte Operationen

Unter *bitorientierten Operationen* soll eine Gruppe von Befehlen zusammengefaßt werden, die es gestattet, einzelne Bits zu verändern bzw. abzufragen.

10.1. Verknüpfungsoperationen

Es können folgende Verknüpfungsoperationen programmiert werden:

logisches UND (N)
logisches ODER (O)
exklusives ODER (X).

Die Befehle arbeiten mit folgenden Operandenlängen:

UND	ODER	Exklusives ODER	Operandenlänge
NR	OR	XR	4 Bytes
N	O	X	4 Bytes
NI	OI	XI	1 Byte
NC	OC	XC	LO-Bytes ($1 \leqq LO \leqq 256$)

Der Bedingungskode wird gestellt.

10.1.1. *Logisches UND (Konjunktion)*

Folgende Befehle stehen zur Verfügung:

```
   1. Verknüpfung im Registerformat
RR-Format  NR   R1,R2
   2. Verknüpfung im Wortformat
RX-Format  N    R1,D2(X2,B2)        Wortgrenze für D2(X2,B2)
   3. Verknüpfung mit Direktzeichen
SI-Format  NI   D1(B1),I2
   4. Verknüpfung im Speicherformat
SS-Format  NC   D1(LØ,B1),D2(B2)
```

Befehlswirkung. Beide Operanden werden bitweise miteinander verknüpft. Es sind alle Bitkombinationen möglich. Das Ergebnis der Verknüpfung steht in den Speicherstellen des ersten Operanden. Die Verknüpfung erfolgt nach folgender Funktionstabelle:

a Erster Operand (vorher)	b Zweiter Operand	$a \wedge b$ Erster Operand (nachher)
1	1	1
1	0	0
0	1	0
0	0	0

Allgemein läßt sich formulieren:

Wenn zweiter Operand $\begin{Bmatrix} 0 \\ 1 \end{Bmatrix}$, wird erster Operand $\begin{Bmatrix} 0 \\ \text{nicht verändert} \end{Bmatrix}$.

Anwendung. Löschen einzelner Bits im Zieloperanden (erster Operand).

Beispiel

Die Bitpositionen 2 und 4 des Feldes ZEIGER sollen ausgeschaltet („0") werden:

```
<ZEIGER>  | beliebig -      |
  vorher  |_|_|_|_|_|_|_|_|
           Ø 1 2 3 4 5 6 7 Bitposition

Befehl:     NI  ZEIGER,B'11Ø1Ø111'
            bzw.
            NI  ZEIGER,X'D7'

<ZEIGER>  |x x Ø x Ø x x x|
  nachher |_|_|_|_|_|_|_|_|
           Ø 1 2 3 4 5 6 7 Bitposition

           x: unverändert
```

10.1.2. *Logisches ODER (Disjunktion)*

Folgende Befehle stehen zur Verfügung:

1. Verknüpfung im Registerformat		
RR-Format	OR	R1,R2
2. Verknüpfung im Wortformat		
RX-Format	O	R1,D2(X2,B2)
3. Verknüpfung mit Direktzeichen		
SI-Format	OI	D1(B1),I2
4. Verknüpfung im Speicherformat		
SS-Format log.	OC	D1(LØ,B1),D2(B2)

Wortgrenze für D2(X2,B2)

Befehlswirkung. Vgl. logisches UND.

Die Verknüpfung erfolgt nach folgender Funktionstabelle:

a Erster Operand (vorher)	b Zweiter Operand	$a \vee b$ Erster Operand (nachher)
1	1	1
1	0	1
0	1	1
0	0	0

Allgemein läßt sich formulieren:

Wenn zweiter Operand $\left\{ \begin{matrix} 1 \\ 0 \end{matrix} \right\}$, wird erster Operand $\left\{ \begin{matrix} 1 \\ \text{nicht verändert} \end{matrix} \right\}$.

Anwendung. Einschalten einzelner Bits im Zieloperanden.

Beispiel

Die Bitpositionen 3 und 5 des Feldes ZEIGER sollen eingeschaltet („1“) werden:

<ZEIGER> vorher | beliebig |
Ø 1 2 3 4 5 6 7 Bitposition

Befehl: OI ZEIGER,B'ØØØ1Ø1ØØ'
bzw.
OI ZEIGER,X'14'

<ZEIGER> nachher | x x x 1 x 1 x x |
Ø 1 2 3 4 5 6 7 Bitposition

x = unverändert

10.1.3. *Exklusives ODER*

Folgende Befehle stehen zur Verfügung:

1. Verknüpfung im Registerformat			
RR-Format	ØR	R1,R2	
2. Verknüpfung im Wortformat			
RX-Format	Ø	R1,D2(X2,B2)	Wortgrenze für D2(X2,B2)
3. Verknüpfung mit Direktzeichen			
SI-Format	ØI	D1(B1),I2	
4. Verknüpfung im Speicherformat			
SS-Format	ØC	D1(LØ,B1),D2(B2)	

Befehlswirkung. Vgl. logisches UND.

Die Verknüpfung erfolgt nach folgender Funktionstabelle:

a Erster Operand (vorher)	b Zweiter Operand	a ⓥ b Erster Operand (nachher)
1	1	0
1	0	1
0	1	1
0	0	0

Allgemein läßt sich formulieren:

Bei $\begin{Bmatrix}\text{Bitgleichheit}\\ \text{Bitungleichheit}\end{Bmatrix}$ wird erster Operand auf $\begin{Bmatrix}0\\1\end{Bmatrix}$ geschaltet.

Anwendung

1. Löschen von Hauptspeicherbereichen auf „binär Null" (X'00').
2. Umstellen von Bitschaltern.

Beispiel

Die Bitpositionen 3 und 5 des Feldes ZEIGER sollen ohne Kenntnis des aktuellen Standes umgeschaltet werden:

<ZEIGER> vorher: Ø Ø 1 1 Ø Ø 1 1
Bitposition: Ø 1 2 3 4 5 6 7

Befehl: XI ZEIGER,B'ØØØ1Ø1ØØ'
bzw.
XI ZEIGER,X'14'

<ZEIGER> nachher: Ø Ø 1 Ø Ø 1 1 1
Bitposition: Ø 1 2 3 4 5 6 7

10.2. Prüfen unter Maske

SI-Format	TM	D1(B1),I2
		↑ zu testende HS-Position → Maske

Befehlswirkung. Alle Bits der durch D1(B1) adressierten Hauptspeicherstelle, denen in der Maske eine „1“ gegenübersteht (korrespondierende Bits), werden geprüft.

Das Ergebnis der Prüfung wird im Bedingungskode gespeichert.

Für die Auswertung der Stellung des Bedingungskodes können folgende Sprungbefehle verwendet (Abschn. 7.) werden:

BO D2(X2,B2) ≙ BC 1,D2(X2,B2)
BM D2(X2,B2) ≙ BC 4,D2(X2,B2)
BZ D2(X2,B2) ≙ BC 8,D2(X2,B2)
BNO D2(X2,B2) ≙ BC 14,D2(X2,B2)

Anwendung. Es soll abgefragt werden, ob die Bitposition 3 des Feldes ZEIGER eingeschaltet ist. Liegt der Fall vor, ist zur Adresse WEITER zu verzweigen, ansonsten soll Bit 3 ausgeschaltet werden.

	:	
	TM	ZEIGER,X'10'
	BO	WEITER
	NI	ZEIGER,X'EF'
	:	
WEITER	:	
ZEIGER	DS	CL1
	:	

Nur diese Bitposition wird ausgewertet
Maske: 00010000

11. Kodeübersetzung, Datenprüfung

Vielfach kommt es vor, daß Daten in eine EDVA eingelesen werden sollen, deren Kodierung nicht mit dem Arbeitskode der EDVA übereinstimmt (Fremdkodes). Solche in Fremdkodes abgelochten Daten können beispielsweise über Lochkarte oder Lochband in die EDVA eingelesen werden. Es kann beispielsweise die Aufgabe gestellt sein, ein im R300-Kode hergestelltes Lochband in die EDVA ROBOTRON 21 einzulesen. Dabei ist eine entsprechende Kodeübersetzung zu programmieren. Speziell dafür kann der Befehl *Umwandeln* (TR) verwendet werden. Eine weitere Aufgabe soll darin bestehen, Datenfelder auf Vorhandensein bestimmter Zeichen zu prüfen und Grenzzeichen von Feldern festzustellen. Dafür ist der Befehl *Umwandeln und Testen* (TRT) anwendbar (Abschnitt 11.2.).

11.1. Kodeübersetzung

SS-Form. Jog.	TR	D1(L0,B1),D2(B2)
		Adr. des zu übersetzenden Feldes (Argumentfeld) — Adr. der Übersetzungstabelle (Funktionsfeld)

Befehlswirkung. TR sucht zu jedem Argumentbyte ein Funktionsbyte heraus und trägt es an die Stelle des Argumentbytes ein.

Bemerkung

1. Mit D1(B1) wird das zu übersetzende Feld (Fremdkode) adressiert. Dieses Feld soll als Argumentfeld bezeichnet werden.

2. Die für die Umwandlung notwendige Übersetzungstabelle (vom Programmierer zu erstellen) wird mit D2(B2) adressiert (Funktionsfeld).
Zum Ablauf der Operationen:
Jedes Byte des Argumentfelds wird als 8-Bit-Dualzahl im Sinne einer relativen Adresse interpretiert. Die Adresse des zugehörigen Funktionsbytes ergibt sich aus der Addition:

Adresse Funktionsbyte := Anfangsadresse Funktionsfeld + relative Adresse

Dieses so gefundene Funktionsbyte wird anstelle des Argumentbytes in das Zielfeld eingetragen. Die Operation ist beendet, wenn das Argumentfeld (Länge L0) mit Bytes des Funktionsfelds gefüllt ist.

Beispiel

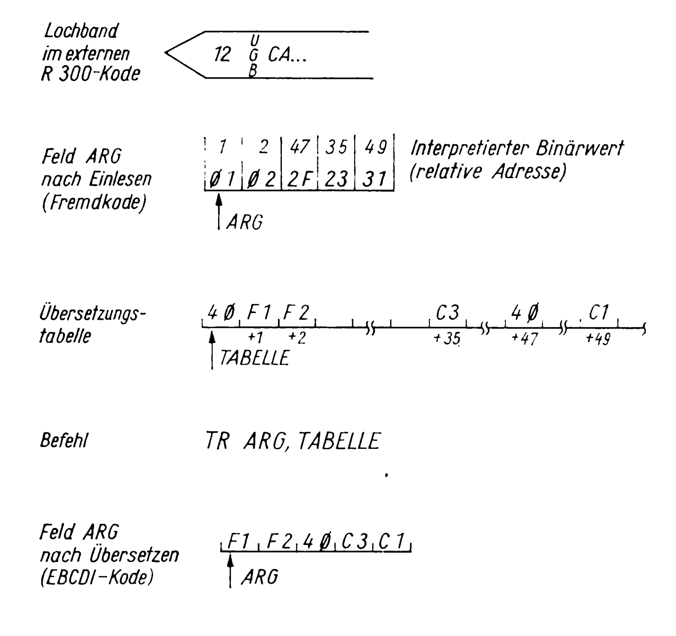
Das Feld ARG soll vom externen R300-Kode in den EBCDI-Kode mit TR übersetzt werden. Zur Vereinfachung werde angenommen, daß nur die Zeichen 0 bis 3 und A bis D sowie Umschaltzeichen auftreten sollen.

Es ist sinnvoll, sich zuerst die entsprechenden Kodeübersichten zu verschaffen (Tafel 3). Danach kann die entsprechende Übersetzungstabelle zusammengestellt werden.

Tafel 3. Kodeübersicht für TR-Befehl

Zeichen	Externer R300-Kode nach dem Einlesen hexadezimal	binär	EBCDI-Kode hexadezimal
0	10	16	F0
1	01	1	F1
2	02	2	F2
3	13	19	F3
A	31	49	C1
B	32	50	C2
C	23	35	C3
D	34	52	C4
UKB	4F	79	40
UGB	2F	47	40

Beim Programmieren muß also jedem vorkommenden Fremdkodezeichen ein äquivalentes Zeichen im EBCDI-Kode auf der entsprechenden Position der Übersetzungstabelle zugeordnet werden.



11.2. Datenprüfung

SS-Formulierung	TRT	D1(LØ,B1),D2(B2)
		Adr. des zu prüfenden Feldes (Argumentfeld) — Adr. der Prüftabelle (Funktionsfeld)

Befehlswirkung. TRT sucht zu jedem Argumentbyte ein Funktionsbyte heraus und untersucht dies auf „ungleich Null". Ein Funktionsbyte „gleich Null" ist für TRT uninteressant; die Operation geht zum nächsten Argumentbyte über.

Bemerkung

1. Mit D1(B1) wird das zu prüfende Feld (Argumentfeld) adressiert.

2. Die notwendige Prüftabelle wird mit D2(B2) adressiert (Funktionsfeld).

3. Die Zuordnung Argumentbyte—Funktionsbyte entspricht dem Befehl TR (Abschn. 11.1.).

4. Ein gefundenes Funktionsbyte „ungleich Null“ löst folgende Schritte aus [6, S. 136]:

Im Register 1 wird rechtsbündig die Adresse des Argumentbytes eingetragen. Die Bits 0 bis 7 im Register 1 bleiben erhalten.

Im Register 2 wird rechtsbündig das Funktionsbyte eingetragen. Die ersten 3 Bytes im Register 2 bleiben erhalten.

Die Abarbeitung wird beendet.

Ist kein Funktionsbyte „ungleich Null“, wird die Operation beendet, wenn das Argumentfeld (Länge L0) vollständig abgearbeitet ist.

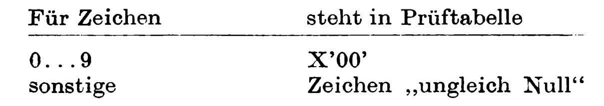
5. Der Bedingungskode wird gestellt.

Beispiel

Das Feld TEST soll in eine arithmetische Operation einbezogen werden. Vor Ausführung dieser Operation soll gesichert sein, daß im Feld TEST nur numerische Zeichen (F0 bis F9) gespeichert sind. Treten andere Zeichen im Feld TEST auf, soll durch das Programm ein entsprechender Hinweis (z. B. über Abfrageeinheit) ausgegeben werden. Für diesen Test kann der Befehl TRT verwendet werden. Dafür ist es notwendig, eine Prüftabelle aufzubauen.

Die entsprechenden Positionen dieser Prüftabelle sind wie folgt zu belegen:

Für Zeichen	steht in Prüftabelle
0...9	X’00’
sonstige	Zeichen „ungleich Null“



Prüftabelle

F F Ø 1 ... C 1 ... E F Ø ... F F

+1 +193 +239 +240 +241 +248 +249 +250 +255

TABELLE

Befehl TRT TEST, TABELLE

Die *Einzelschritte des Befehls* sind

1. Prüfen ⟨TABELLE + 241⟩: Belegung X’00’, nächstes Argumentbyte prüfen
2. Prüfen ⟨TABELLE + 242⟩: Belegung X’00’, nächstes Argumentbyte prüfen
3. Prüfen ⟨TABELLE + 193⟩: Belegung „ungleich Null“.

Wirkung

Bedingungskode wird auf „1“ gestellt.
Adresse des Argumentbytes (TEST + 2) wird in Register 1 eingetragen.
Funktionsbyte (X'C1') wird in Register 2 eingetragen.
Befehl wird abgebrochen.

12. Druckaufbereitung

Durch die Operation *Druckaufbereitung* ist es möglich, daß eine gepackte Zahl mit Hilfe einer Schablone ein durch Punkt, Komma, Minuskennzeichen usw. aufbereitetes Druckbild erhält. Dadurch kann die Druckliste jede beliebige Form annehmen.
Der Befehl für Druckaufbereitung lautet:

SS-Form.log.	ED	D1(L0,B1),D2(B2)
		Adr. der Schablone / Adr. der einzuordnenden Zahl (Datenfeld)

Befehlswirkung. Die durch D2(B2) adressierte gepackte Dezimalzahl wird unter Steuerung der durch D1(B1) adressierten Schablone zum Druck aufbereitet und dabei entpackt.

12.1. Steuerung durch Schablone

Die Schablone kann beliebige Zeichen enthalten. Es ist sinnvoll, die Zeichen in *4 Klassen* einzuteilen:

Klasse	*Bedeutung*	*Symbol*	*Kodierung*
0	*Ziffernauswahlzeichen*	▽	X'20'
1	*Bedeutungsbeginnzeichen*	▽	X'21'
2	*Feldtrennungszeichen*	▽	X'22'
3	*sonstige druckbare Zeichen*		

Bei der Operation ED wird jedes Zeichen der Schablone auf seine Klassenzugehörigkeit hin untersucht. In Abhängigkeit davon werden verschiedene Funktionen ausgelöst. Das erste Zeichen der Schablone wird als Füllzeichen gewertet. Die Schablonenzeichen werden entweder durch das Füllzeichen oder eine auf der Adresse D2(B2) stehende Ziffer ersetzt, oder sie bleiben in ihrer Form erhalten.

Bemerkung

1. Bei Beginn der Druckaufbereitung ist Nullenunterdrückung wirksam.
2. Die Nullenunterdrückung wird für das folgende Zeichen aufgehoben durch

   eine Ziffer 1 bis 9 auf der Adresse D2(B2)
   das Bedeutungsbeginnzeichen in der Schablone.

3. Die Nullenunterdrückung wird wieder wirksam durch
   eine Ziffer in Verbindung mit einem positiven Vorzeichen auf der Adresse D2(B2)
   das Feldtrennungszeichen in der Schablone.

4. Ist die Nullenunterdrückung wirksam, ersetzt das erste Zeichen der Schablone (Füllzeichen)
   die führenden Nullen im Datenfeld
   das in der Schablone vorhandene Zeichen.

5. Das Füllzeichen bleibt unverändert erhalten.

6. Das Feldtrennungszeichen wird durch das Füllzeichen ersetzt.

12.2. Anwendungen

Es erscheint sinnvoll, vor den Anwendungen einige Hinweise für die Programmierung zu geben.

1. In Druckaufbereitungen sollten nur Zahlen einbezogen werden, in denen Vornullen beseitigt werden sollen (also nicht konstante Werte wie Artikelnummer, Kundennummer usw.).

2. Das Listenbild bildet die Grundlage für den Aufbau der Schablone.

3. Zur besseren Übersicht empfiehlt es sich, die Schablone mit den auf Seite 92 dargestellten Symbolen (Hilfsmittel) aufzuschreiben.

4. Diese Symbole dürfen nicht in das Programmformular eingetragen werden, da sie im Arbeitskode nicht vereinbart sind.

5. Anstelle der Symbole sind bei der Definition der Schablone (als Konstante definieren) die entsprechenden Hexadezimalzahlen zu verwenden.

6. Die Schablone muß genauso viele X'20'- und X'21'-Zeichen enthalten, wie im Datenfeld Ziffern (ohne Vorzeichen) vorhanden sind (gepacktes Format beachten).

7. Im Datenfeld, auf der Adresse D2(B2), muß eine gepackte Dezimalzahl stehen (sonst Programmausnahmeunterbrechung).

8. Vor dem Befehl ED muß die Schablone in das Zielfeld, Adresse D1(B1), übertragen werden.

9. Der Befehl ED ist beendet, wenn alle LO-Zeichen ($1 \leqq LO \leqq 256$) verarbeitet sind.

10. Die Verarbeitung erfolgt von links nach rechts.

11. Bei ED wird der Bedingungskode gestellt.

12.2.1. *Vorzeichenbehandlung*

Es werden alle Zeichen der Schablone, die rechts vom letzten Ziffernauswahlzeichen (X'20') oder Bedeutungsbeginnzeichen (X'21') stehen, als Minuskennzeichnung aufgefaßt. In Abhängigkeit des Vorzeichens im Datenfeld, Adresse D2(B2), wird bei positivem Vorzeichen die Minuskennzeichnung unterdrückt; bei negativem Vorzeichen bleibt sie erhalten.

Beispiel

Gefordertes Druckbild ZZZ9,99ƀ MINUS

<Datenfeld> Ø Ø , Ø 4 , 3 2 , 1 +
↑ DATEN

Schablone ƀ▽ ▽ ▽ ∀ ▽, ▽ ▽ ƀ MINUS

gedruckte Zahl 43,21

Bemerkung: Z = Vornullen sollen unterdrückt werden!

Die Programmierung sieht wie folgt aus:

```
        :
        <DATEN> errechnen
        :
        MVC   AUSG(15),SCHABL        Schablone übertragen
        ED    AUSG(15),DATEN         Druckaufbereitung
        :
SCHABL  DC    X'4Ø2Ø2Ø2Ø212Ø'  )
        DC    C','             ) Definition der Schablone
        DC    X'2Ø2Ø4Ø'        )
        DC    C'MINUS'         )
AUSG    DS    CL...
DATEN   DS    CL4
        :
```

Wenn im Feld DATEN die gleiche Ziffernfolge, allerdings mit negativem Vorzeichen, gespeichert wäre, würde mit der gleichen Befehlsfolge 43,21 MINUS gedruckt.

12.2.2. *Aufbereiten mehrerer Dezimalzahlen*

Es besteht die Forderung, in einem Druckaufbereitungsbefehl mehrere Zahlen aufzubereiten. Das im Abschn. 12.1. eingeführte Feldtrennungszeichen wird hierbei angewendet.

Beispiel

Gefordertes Druckbild ZZ9,99ƀ-ƀƀ ZZ9,99ƀ-

<Datenfeld> Ø 4 , 3 2 , 1 - , Ø Ø , 1 2 , 3 +
↑ DATEN

Schablone ƀ▽∀▽,▽▽ƀ-⊽ƀ▽∀▽,▽▽ƀ-

gedruckte Zahl 4 3 , 2 1 - 1 , 2 3

Bemerkungen: – VZ = Vorzeichen, gedruckt wird { – / ƀ (nichts gedruckt)
– ⊽ schaltet Nullenunterdrückung wieder ein

Programmiert wird wie folgt:

```
         :
         MVC   AUSG(19),SCHABL        Schablone übertragen
         ED    AUSG(19),DATEN         Druckaufbereitung
         :

SCHABL   DC    X'4Ø2Ø212Ø'
         DC    C','
         DC    X'2Ø2Ø4Ø'
         DC    C'-'
         DC    X'224Ø2Ø212Ø'
         DC    C','
         DC    X'2Ø2Ø4Ø'
         DC    C'-'
AUSG     DS    CL...
DATEN    DS    CL6
         :
```

13. Ausblick

Der vorliegende Band befaßt sich vorwiegend mit internen Operationen. E/A-Techniken wurden nicht betrachtet. Ebenso ist auf die Wirkungsweise der Steuerprogramme nicht eingegangen worden. Diese Themen werden in den Bänden RA 141 und RA 142 der REIHE AUTOMATISIERUNGSTECHNIK von den gleichen Autoren behandelt.

Zusammenfassenden Beispielen und komplexen Übungen ist der Band „Dateibehandlung; Programmierbeispiele“ (RA 143) gewidmet.

14. Lösungen

Ü. 2.1. a) gepackt: 0000 0001 0000 1101
und 0100 0000 0000 1100

b) ungepackt: 1111 0001 1111 0010

Ü. 2.2. a) 21 = X'15'

b) 16 = X'10'

c) 37 = X'25'

Ü. 4.1. Fehlerhafte Symbole: BEZ1.1 und TEILE.-NR.

Ü. 4.2. Bereichsdefinition

```
EIN        DS   OCL50
TEILENR    DS   CL7
BEZEICH    DS   CL39
MENGE      DS   CL4
```

Ü. 4.3. Bereichsdefinition

```
ZWSP       DS   D
```

Ü. 4.4. Konstantendefinitionen

```
        a)  DC   C'UNGUELTIGER WERT'
            DC   C'1.9.1972'
        b)  DC   P'-70'
            DC   Z'60'
        c)  DC   F'37'
            DC   H'-1'
```

Ü. 5.

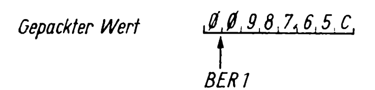
```
               CLC  KNR,=C'10000'
               .
               .
               .
KNR            DS   CL5
```

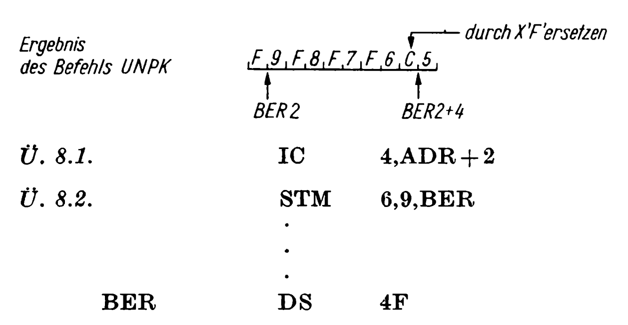
Bemerkung. Die Auswertung des Vergleichs erfolgt durch einen Sprungbefehl.

Ü. 6.

```
               UNPK  BER2,BER1
               MVZ   BER2+4(1),=X'FO'
               .
               .
               .
BER1           DS    CL4
BER2           DS    CL5
```

Das Ergebnis der Operation sieht wie folgt aus:





Ü. 8.1. IC 4,ADR+2

Ü. 8.2. STM 6,9,BER

```
                  STM    6,9,BER
                  .
                  .
                  .
         BER      DS     4F
```

Bemerkung

R6 nach BER

R7 nach BER+4 usw.

Ü. 8.3. Zuerst ist der Dividend im Registerpaar [R6, R7] bereitzustellen (SRDA). Als Divisor kann das Literal =F'10' verwendet werden.

```
                  SRDA   6,32
                  D      6,=F'10'
                  ST     7,QUOT
                  .
                  .
                  .
         QUOT     DS     F
```

Literaturverzeichnis

[1] *Eilitz, M.:* Betriebssysteme der 3. Rechnergeneration. Rechentechnik/Datenverarbeitung 8 (1971) 3, S. 9.

[2] *Paulin, G.:* Grundzüge des Programmierens. RA 131. Berlin: VEB Verlag Technik 1972.

[3] —: EDVA ROBOTRON 21 (Aufbau, Arbeitsweise, E/A-Geräte, Plattenbetriebssystem u. a.). Rechentechnik/Datenverarbeitung 8 (1971) 11/12, S. 4—71.

[4] —: Lehrhilfen Programmierung ROBOTRON 21. VEB KOMBINAT ROBOTRON, Zentralvertrieb, Schulungszentrum.

[5] —: Assemblerhandbuch EDV-System ROBOTRON 21. VEB KOMBINAT ROBOTRON.

[6] —: Programmierhandbuch EDV-System ROBOTRON 21, Teil 1 und 2. VEB KOMBINAT ROBOTRON.

[7] *Germain, C. B.:* Das Programmierhandbuch der IBM/360. München: Hanser Verlag 1969.

[8] —: ESER-Modell R20 in die Produktion übergeleitet. Rechentechnik/Datenverarbeitung 9 (1972) 8, S. 1.

[9] —: EDVA R30 — Eine sowjetisch-polnische Gemeinschaftsentwicklung innerhalb der ESER-Reihe. Rechentechnik/Datenverarbeitung 9 (1972) 9, S. 35—37.

[10] —: DDR-Standard TGL 23 207: 7-Bit-Kode.

[11] —: DDR-Standard TGL 22 451: Datenfluß- und Programmablaufpläne.

Abkürzungsverzeichnis

1. Operationskodes

A	add	Addition Wort
AH	add halfword	Addition Halbwort
AR	add register	Addition Register
AP	add decimal	Addition Dezimal
BAL	branch and link	Verzweigen und Laden Folgeadresse (RX)
BALR	branch and link	Verzweigen und Laden Folgeadresse (RR)
BC	branch on condition	Verzweigen nach Bedingung (RX)
BCR	branch on condition	Verzweigen nach Bedingung (RR)
BCT	branch on count	Verzweigen nach Zählwert (RX)
BCTR	branch on count	Verzweigen nach Zählwert (RR)
BXH	branch on index high	Verzweigen bei Index „größer"
BXLE	branch on index low or equal	Verzweigen bei Index „kleiner oder gleich"
C	compare	Vergleich Wort
CCW	channel command word	Kanalkommandowort
CH	compare halfword	Vergleich Halbwort
CL	compare logical	Vergleich logisch
CLC	compare logical characters	Vergleich logisch Zeichen
CLI	compare logical immediate	Vergleich logisch Direktzeichen
CR	compare register	Vergleich Register
CVB	convert to binary	Konvertieren ins Binäre
CVD	convert to decimal	Konvertieren ins Dezimale
D	divide	Division Wort
DC	define constant	definiere Konstante
DS	define storage	definiere Speicher
DR	divide register	Division Register
DP	divide decimal	Division Dezimal
ED	edit	Druckaufbereiten
IC	insert character	Einsetzen Zeichen
L	load	Laden Wort
LA	load address	Laden Adresse
LH	load halfword	Laden Halbwort
LM	load multiple	Laden mehrfach
LR	load register	Laden Register
M	multiply	Multiplikation Wort
MH	multiply halfword	Multiplikation Halbwort
MR	multiply register	Multiplikation Register
MP	multiply decimal	Multiplikation Dezimal
MVC	move characters	Übertragen Zeichen
MVI	move immediate	Übertragen Direktzeichen
MVN	move numerics	Übertragen numerischen Teil
MVO	move with offset	Übertragen mit Absetzen
MVZ	move zones	Übertragen Zonenteil
N	and	logisches UND (Wort)
NC	and characters	logisches UND (Zeichen)
NI	and immediate	logisches UND (Direktzeichen)

NR	and register	logisches UND (Register)
O	or	logisches ODER (Wort)
OC	or characters	logisches ODER (Zeichen)
OI	or immediate	logisches ODER (Direktzeichen)
OR	or register	logisches ODER (Register)
PACK	pack	Packen
S	subtract	Subtraktion Wort
SH	subtract halfword	Subtraktion Halbwort
SLA	shift left single	Linksverschiebung einfach
SLDA	shift left double	Linksverschiebung doppelt
SP	subtract decimal	Subtraktion dezimal
SR	subtract register	Subtraktion Register
SRA	shift right single	Rechtsverschiebung einfach
SRDA	shift right double	Rechtsverschiebung doppelt
ST	store	Speichern Wort
STC	store character	Speichern Zeichen
STH	store halfword	Speichern Halbwort
STM	store multiple	Speichern mehrfach
TM	test under mask	Prüfen unter Maske
TR	translate	Übertragen
TRT	translate and test	Übertragen und Testen
UNPK	unpack	Entpacken
USING	use base address register	Zuweisen Basisregister
X	exclusive or	exklusives ODER (Wort)
XC	exclusive or characters	exklusives ODER (Zeichen)
XI	exclusive or immediate	exklusives ODER (Direktzeichen)
XR	exclusive or register	exklusives ODER (Register)
ZAP	zero and add	Löschen und Addieren

2. Sonstige Abkürzungen

B1	Basisregisternummer für ersten Operand (0 bis 15)
B2	Basisregisternummer für zweiten Operand (0 bis 15)
DOS	Plattenbetriebssystem (disk operating system)
DOS/ES	Betriebssystem im ESER
D1	Verschiebung (displacement) des ersten Operanden (0 bis 4095)
D2	Verschiebung des zweiten Operanden (0 bis 4095)
GSE	Gerätesteuereinheit
GSS	Großraumspeichersteuergerät
I2	Direktzeichen (immediate)
L0	Länge in Bytes (1 bis 256)
L1	Länge erster Operand in Bytes (1 bis 16)
L2	Länge zweiter Operand in Bytes (1 bis 16)
M1	Maske
OS/ES	Betriebssystem im ESER
PSW	Programmstatuswort
R1, R2	Nummer allgemeiner Register (0 bis 15)
RR-Format	Register—Register
RS-Format	Register—Speicher
RX-Format	Register—Speicher
SI-Format	Speicher—Direktzeichen
SS-Format	Speicher—Speicher
SIF	Standardanschlußbild (standardinterface)

REIHE AUTOMATISIERUNGSTECHNIK

• Zur Zeit lieferbare oder in Vorbereitung befindliche Bände

RA 51 *Bode:* Lochkartentechnik
•RA 52 *Paulin:* Kleines Lexikon der Rechentechnik und Datenverarbeitung
RA 53 *Greif:* Meßwert-Registriertechnik
•RA 54 *Jeschke:* Kleines Lexikon der Betriebsmeßtechnik
•RA 55 *Töpfer* u. a.: Pneumatische Bausteinsysteme der Digitaltechnik
•RA 56 *Weller:* Regelung von Dampferzeugern
RA 57 *Mütze:* Numerisch gesteuerte Werkzeugmaschinen
RA 58 *Heimann:* Radionuklide in der Automatisierungstechnik
RA 59 *Fuchs/Weller:* Mehrfachregelungen
RA 60 *Queisser:* Instandhaltung von Automatisierungsanlagen
RA 61 *Peschel:* Statistische Methoden in der Regelungstechnik
•RA 62 *Töpfer* u. a.: Pneumatische Steuerungen
•RA 63 *Kochan/Strempel:* Programmgesteuerte Werkzeugmaschinen
RA 64 *Brenk/Eichner:* Integrierte Datenverarbeitung
•RA 65 *Gensel:* Zerstörungsfreie Prüfverfahren
RA 66 *Worgitzki:* Elektrisch-analoge Bausteine der Antriebstechnik
•RA 67 *Kerner/Grützner:* Praxis der ALGOL-Programmierung
RA 68 *Pankalla:* Aufbau und Einsatz von Prozeßrechenanlagen
RA 69 *Timpe:* Ingenieurpsychologie und Automatisierung
•RA 70 *Böhme:* Periphere Geräte der digitalen Datenverarbeitung
RA 71 *Dutschke/Grebenstein:* BMSR-Einrichtungen in ex-gefährdeten
RA 72 *Müller:* Automatisierungsanlagen [Betriebsstätten
•RA 73 *Paulin:* FORTRAN — Kodierung von Formeln
•RA 74 *Paulin:* FORTRAN — Datenbeschreibung/Unterprogrammtechnik
RA 75 *Gottschalk:* Darstellungen und Symbole der Automatisierungstechnik
•RA 76 *Hart:* Kontinuierliche Flüssigkeitsdichtemessung
RA 77 *Börnigen:* Elektronische Datenverarbeitungsanlage ROBOTRON 300
RA 78 *Krebs:* Rechner in industriellen Prozessen
RA 79 *Böhme/Born:* Programmierung von Prozeßrechnern
•RA 80 *Lemgo/Tschirschwitz:* Programmierung des R 300 — Zentraleinheit
•RA 81 *Lemgo/Tschirschwitz:* Programmierung des R 300 — Peripherie
RA 82 *Mikutta* u. a.: Bauelemente der Industriepneumatik
RA 83 *Dörband* u. a.: Praxis der FORTRAN-Programmierung — Grundstufe
RA 84 *Dörband* u. a.: Praxis der FORTRAN-Programmierung — Oberstufe
•RA 85 *Kautsch:* Elektronenstrahl-Oszillografie
RA 86 *Bürger/Leonhardt:* Lochbandtechnik
RA 87 *Trognitz/Wegner:* Physiologische Arbeitsgestaltung
RA 88 *Kadow/Kerner:* Programmieranweisung ZRA 1
RA 89 *Eube/Illge:* Membran-Stellventile
•RA 90 *Woschni:* Meßfehler
RA 91 *Biener/Suschke:* Praxis des analogen Rechnens
•RA 92 *Wahl:* Grundlagen der Elektronik
RA 93 *Bürger:* Informationsspeicher
•RA 94 *Hartmann:* Aufgabensammlung Regelungstechnik
•RA 95 *Ludwig:* Regelung von Dampf- und Gasturbinenanlagen
•RA 96 *Draeger:* Automatisierung und Berufsbildung in der DDR
•RA 97 *Strejc:* Dimensionierung stetiger linearer Regelkreise für die Praxis
•RA 98 *Woschni:* Information und Automatisierung
•RA 99 *Meuche:* Komplexe automatische Informationsverarbeitungssysteme
•RA 100 *Peschel:* Kybernetische Systeme

REIHE AUTOMATISIERUNGSTECHNIK

* Zur Zeit lieferbare oder in Vorbereitung befindliche Bände

RA 101 *Beichelt:* Zuverlässigkeit und Erneuerung
*RA 102 *Franke:* Abtastregelkreise mit Relaisreglern
*RA 103 *Paulin:* ALGOL-Training
*RA 104 *Reinecke/Trenkel:* Aut. Zeichenerkennung — Techn. Grundlagen
*RA 105 *Reinecke/Trenkel:* Aut. Zeichenerkennung — Geräte und Anwendung
RA 106 *Otto/Peschel:* Korrelations- und Spektralanalyse
RA 107 *Bittner:* Pneumatische Meßumformer und Regler
*RA 108 *Pabst:* Operationsverstärker
*RA 109 *Hartmann:* Praxis der elektronischen Datenverarbeitung
*RA 110 *Kerner:* Kurze Einführung in ALGOL 60
*RA 111 *Ober/Schumann:* Programmierung des R 300 — Standardprogramme
RA 112 *Schubert:* Gleichspannungsverstärker
*RA 113 *Sydow:* Elektronisches Hybridrechnen
RA 114 *Müller:* Verfahrenstechnik und Automatisierung
*RA 115 *Brack:* Dynamische Modelle verfahrenstechnischer Prozesse
*RA 116 *Leupold/Lötzsch:* Programmierung des C 8205 — Maschinenkode
*RA 117 *Ludwig:* Anlagenautomatisierung
RA 118 *Stempell:* Einführung in PL/1
*RA 119 *Grützner:* PL/1-Training
*RA 120 *Oberländer:* Datenerfassung in der Stückgut- und Chargenfertigung
*RA 121 *Nitzsche:* Magnetische und elektrische zerstörungsfreie Prüfung
*RA 122 *Paulin:* FORTRAN-Training
*RA 123 *Oberst:* Kombinationsschaltungen
*RA 124 *Möller:* Elektrische Klimaregelung
*RA 125 *Peschel:* Statistische Kennwertermittlung
*RA 126 *Leupold/Lötzsch:* Programmierung des C 8205 — Maschinenkode-orientierte Interpretiersysteme
*RA 127 *Herold:* Physikalisch-technische Grundlagen elektron. Verstärker
*RA 128 *Jahn/Kühne/Vahle:* Prozeßmodellierung
RA 129 *Wolf:* Thyristoren
*RA 130 *Bürger:* Automatisierung der technischen Produktionsvorbereitung
*RA 131 *Paulin:* Grundzüge des Programmierens
*RA 132 *Elzner:* Datensicherung
*RA 133 *Hesse/Zapf:* Automatisches Fügen
*RA 134 *Schreiter u. a.:* SYMAP (B)
*RA 135 *Schreiter u. a.:* SYMAP (PS) und (DB)
RA 136 *Franke:* Datenübertragung
*RA 137 *Peschel:* Statistische Modellbildung
RA 138 *Kabisch:* Anwendung von Thyristoren
*RA 139 *Müller/Müller:* Fortschritte der Stelltechnik
*RA 140 *Kern/Ober/Schumann:* ESER-Programmierung im Betriebssystem DOS/ES — Systembeschreibung, Assemblersprache —
*RA 141 *Kern/Ober/Schumann:* ESER-Programmierung im Betriebssystem DOS/ES — Ein- und Ausgabetechnik, Arbeit mit Wechselplattenspeichern —
*RA 142 *Kern/Ober/Schumann:* ESER-Programmierung im Betriebssystem DOS/ES — Steuerprogramme, Bibliotheksführung —
*RA 143 *Kern/Ober/Schumann:* ESER-Programmierung im Betriebssystem DOS/ES — Dateibehandlung; Programmierbeispiele —
*RA 144 *Kretschmer:* Maschinelles Nachweisen von Dokumenten
*RA 145 *Herold:* Schaltungstechnik und Aufbau elektronischer Verstärker
*RA 146 *Hesse:* Zuverlässigkeitsarbeit
*RA 147 *Schreiter* u. a.: SYMAP-Praxis
*RA 148 *Grützner/Priem:* PL/1 Daten — Anweisungen — Programme
*RA 149 *Grützner/Priem:* PL/1 Blöcke — Strukturen — Prozeduren
*RA 150 *Kern/Schubert:* Übersicht über das EDV-System ROBOTRON ES1040